**1. Многоуровневая организация вычислительной системы.**

ВС организуется как многоуровневая система, в которой каждому уровню соответствуют определенные компоненты и средства (языки) описания. Введение нескольких уровней абстракции позволяет справиться со сложностью ВС. Выделяют уровни языков прикладного программирования и моделирования, ассемблера, ОС, архитектуры системы команд, микроархитектуры, цифровой логический уровень, физический уровень.



**2. Физический уровень ЭВМ. Технологии элементной базы ЭВМ.**

Физический уровень в традиционных ЭВМ - это уровень электронных схем. Формальным средством описания на данном уровне являются уравнения для токов и напряжений в электроцепях.

Потенциальные элементы ЭВМ на физическом уровне различают по схемотехническому признаку - способу соединения транзисторов, диодов и резисторов между собой в пределах одной схемы типового базового элемента. Принято считать, что совокупность элементов с общим признаком построения образует вид схемной логики.

Различают следующие виды схемной логики потенциальных элементов:

диодно-транзисторную (ДТЛ)

интегральную инжекционную (ИИЛ)

транзисторно-транзисторную (ТТЛ) и её модификации с диодами Шотки (ТТЛШ)

эмиттерно-связанную (ЭСЛ)

МОП-транзисторную (p-МОП, n-МОП и КМОП)

логику на основе арсенида галлия (AsGa)

другие.

В прошлом предпочтение отдавалось схемотехнике на основе биполярных транзисторов (ТТЛШ, ЭСЛ), которые обладают меньшим временем задержки переключения по сравнению с полевыми транзисторами. В настоящее время физический уровень ЭВМ реализуется на основе полевых транзисторов (КМОП). Схемы КМОП характеризуются относительной простотой изготовления, высокой степенью интеграции (компактностью), большой нагрузочной способностью и помехоустойчивостью, малым потреблением энергии. КМОП-технология является основой современных микропроцессорных систем. Недостатком МОП-структур является низкое быстродействие по сравнению с ТТл и ЭСЛ.

**3. Физические основы перспективной вычислительной техники.**

Квантовые компьютеры

Идея квантовых вычислений, как нового способа осуществления параллельных вычислительных процессов, стала активно обсуждаться после опубликования в 1982 году стать Ричарда Фейнмана, в которой отмечалось, что неопределенные и недетерминированные квантовые физические процессы можно использовать для разработки компьютеров нового типа, вычисляющих на основе “квантовых алгоритмов”.

В середине 80-х появились работы Дойча, Яо, Шора, Беристейна и Вазирани, в которых были построены формальные модели квантового компьютера и квантовых вычислений. Ускорение процесса решения задач на квантовом компьютере связывают с квантовой природой кубитов, которые представляют собой квантовые частицы, имеющие два базовых состояния. В каждый данный момент времени кубит кодирует одновременно “1” и “0” (в этом случае говорят, что кубит находится в состоянии квантовой суперпозиции).

Измерение значения кубита носит вероятностный характер: например, до проведения измерения значением кубита может быть “1” с вероятностью 20% и одновременно “0” с вероятностью 80%. Измерение, производимое над квантовым объектом, оказывает на него воздействие и вынуждает перейти (коллапсировать) в одно из состояний. Свойство квантовых частиц быть одновременно во многих состояниях, называемое квантовым параллелизмом, используется в квантовых вычислениях.

Вычисления в квантовой модели осуществляются одновременно со всеми базисными состояниями, на которые разложен кубит. Так как для одного кубита используются два базисных состояния, то при работе с одним кубитом вычисления одновременно производятся с двумя квантовыми состояниями. В многокубитовой системе вычисления одновременно осуществляются с экспоненциальным числом квантовых состояний. Для L-кубитовой системы при разложении на суперпозицию базисных состояний должен определяться новый базис, в котором используется 2^L базисных состояний.

Достаточно долго оставалось неясным, можно ли использовать гипотетическую вычислительную мощь квантового компьютера для ускорения решения практических задач. С появление в 1994 году статьи Питера Шора, в которой был предложен эффективный квантовый алгоритм вычисления дискретного логарифма, ситуация изменилась. Шор разработал алгоритм, который позволил проводить быструю факторизацию больших чисел. По сравнению с лучшим из существующих на сегодня классических методов квантовый алгоритм Шора дает многократное ускорение вычислений, причем, чем длиннее факторизируемое число, тем больше выигрыш в скорости.

До появления работы Шора задача разложения целых чисел на простые множители и, как следствие, вычисление дискретного логарифма, считалась настолько сложной задачей, что разработчики криптографических систем основывались на предположении о том, что вычислить используемый в криптографии дискретный логарифм за приемлемое время невозможно, если модуль - достаточно большое простое число. Таким образом, квантовый алгоритм факторизации, позволяющий производить разложение n-значного числа на простые множители за время, полиномиально зависящее от n, то есть с экспоненциальным ускорением по сравнению с самыми мощными классическими алгоритмами, стал одним из основных стимулов для интенсивного развития квантовых методов вычисления.

Однако, для того чтобы квантовый компьютер дал ощутимый выигрыш в производительности по сравнению с современными компьютерами и оправдал затраченные на его создание средства, регистр должен содержать не менее 1000 кубитов. В настоящее время квантовые компьютеры, которые имели бы квантовые регистры, включающие не менее 1000 кубитов, а также удовлетворяли ряду других требований, в частности, вытекающих из условия помехозащищенности квантовых вычислений, существуют только теоретически. Предполагается, что большого числа кубитов в квантовом регистре можно будет достигнуть лишь в твердотельном исполнении квантовых компьютеров, работающих в условиях низких температур. Существенным препятствием на пути построения квантовых компьютеров является то, что внешняя среда за довольно короткое время, называемое временем декогерентности, разрушает квантовое состояние, поэтому при выборе той или иной технологии прежде всего принимается во внимание число вычислительных шагов, которые можно успеть совершить, пока декогерентность не разрушит процесс. Другой фундаментальной проблемой квантовых компьютеров является измерение состояния кубитов после вычислений, так как нельзя получить информацию о кубите, не повлияв на его состояние при измерении.

Оптоэлектронные и оптические вычислительные системы

Оптоэлектронные системы интегрируют преимущества оптики и электроники. Эти системы используются не только для хранения и передачи информации, но и для эффективной обработки данных. Оптоэлектронные вычислительные машины могут применяться при решении задач обработки радиоэлектронных сигналов, спектрального и корреляционного анализа, распознавания образов. Оптоэлектронные системы способны выполнять преобразования Фурье двухмерного сигнала существенно быстрее электронных компьютеров.

В случае, когда требуется произвести преобразование функции, непрерывно изменяющейся во времени, для ее представления можно применить дискретизацию, однако с помощью цифровой технологии не удается добиться непрерывного преобразования непрерывных во времени сигналов. Здесь нужны аналоговые методы, а аналоговая оптическая обработка больших массивов выполняется быстрее, чем аналоговая электронная.

Оптические методы позволяют производить как аналоговую, так и дискретную обработку информации. Интерес к дискретной оптической обработке информации связан с необходимостью преодоления проблем, с которыми сталкивается аналоговая оптическая вычислительная техника. В дискретных оптических вычислениях световые сигналы используются для осуществления операций цифровой логики и нацелены на класс приложений, которые могут быть ускорены за счет применения оптоэлектроники.

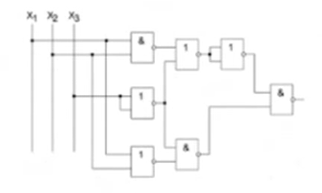
Важный шаг состоит в разработке полностью оптической или оптоэлектронной технологии вентилей, которая позволила бы воспроизвести функциональность традиционной электроники, сохранив преимущества оптики в передаче сигналов. Поскольку по степени интеграции и сложности выполнения операций цифровые оптические логические элементы продолжают значительно уступать твердотельным электронным решениям, применение цифровых оптических технологий в ближайшем будущем будет выгодно только в двух случаях: когда приложение проявляет высокую степень параллелизма при простоте вычислительных операций; когда эффективное выполнение приложения требует создания обширной и сложной системы соединений между массивами вычислительных элементов.

Аналого-цифровые оптические вычисления выполняются с привлечением гибридных технологий и систем, которые могут обрабатывать оба вида данных. В качестве примера можно привести оптический ассоциативный процессов - систему, которая имеет доступ и может обрабатывать записи по содержанию полей записи, а не по адресу. В отличие от электронных ассоциативных систем он способен обрабатывать как цифровые, так и аналоговые данные, а также осуществлять ассоциативную выборку информации, хранящейся в голографической памяти.

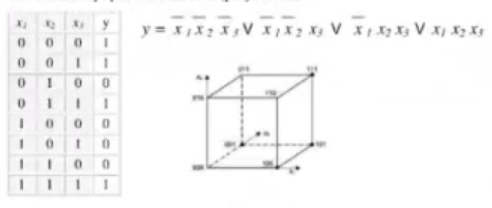
Молекулярная наноэлектроника

Многообещающим является создание наноструктур, в которых роль функциональных элементов выполняют отдельные молекулы. В перспективе это может позволить реализовать параллельные вычислительные процессы в биологических объектах (молекулярная наноэлектроника). Однако у молекулярных компьютеров есть существенный недостаток: для реализации вычислений необходимо каждый раз переводить каждую молекулу в её первоначальное состояние, что требует значительных временных затрат. Другая проблема связана с вводом и выводом информации для пользователя. Так как информационные процессы в молекулярной электронике имеют специфический характер, существуют проблемы сопряжения вычислительных устройств и преобразования сигналов из одной формы в другую.

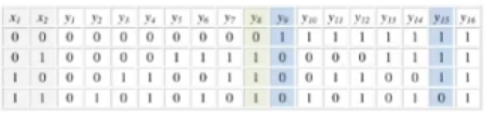
**4. Уровень логических элементов. Логические функции и логические схемы.**

Цифровая логическая схема - это схема, в которой используется только два логических значения 0 и 1. В комбинационной схеме сигнал на выходе схемы зависит только от входных сигналов. Для анализа и синтеза КС используется математический аппарат булевой алгебры. Пример: 

Функция y=f(x1,x2,...xn) от n переменных называется логической (или булевой) функцией, если сама функция и любой из ее аргументов могут принимать значение только из множества {0.1}. Так как область значений булевых функций полностью совпадает с областью определения, в булевой алгебре легко реализуется принцип суперпозиции, то есть использование значений функций в качестве аргумента. Для описания логический функций используются аналитический, графический и табличный способы.



Областью определения булевой функции является множество всевозможных n-мерных двоичных наборов, количество которых равно 2^n. Общее количество функций от n переменных равно (2^2)^n. Общее число функций от двух переменных равно 16.



Среди шестнадцати функций от двух переменных существую две, которые являются функционально полными:



1 – стрелка Пирса

2 – штрих Шеффера

Система булевых функций называется функционально полной, если для любой булевой функции в данной система от n переменных можно построить равную ей булеву функцию, представляющую собой результат суперпозиции переменных и функций системы, взятой в конечном экземпляре каждая.

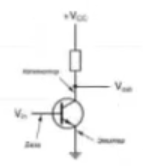
Алгебра над множеством логических функций с двумя бинарными операциями (конъюнкция и сложение по модулю два) называется алгеброй Жегалкина.

Полученная формула, имеющая вид суммы произведений, называется полиномом Жегалкина.

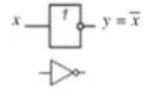
Основные типы комбинационных схем

Цифровая схема – это схема, в которой используется только два логических значения: 0 и 1.

Современная цифровая логика основывается на том, что транзистор может работать как очень быстрый двоичный переключатель.



Это схема является инвентором и на цифровом логическом уровне обозначается в виде логического элемента:



Наиболее распространенными комбинационными устройствами являются дешифраторы, шифраторы, семисегментные дешифраторы, мультиплексоры и демультиплексоры, арифметические сумматоры и арифметико-логические устройства (АЛУ).

**5. Задачи анализа и синтеза комбинационных схем. Последовательность синтеза комбинационных схем.**

Задача анализа логических схем состоит в нахождении функции реализуемой схемой. Задача синтеза логических схем формулируется следующим образом: при заданных входных аргументах, известной входной функции и ограничениях, накладываемых на использование логических элементов, необходимо синтезировать логическое устройство, которое реализует эту функцию.

Последовательность синтеза логических схем:

Задаются требования к схеме

Составляется таблица истинности

По таблице истинности находится совершенная дизъюнктивная форма функции (СДНФ)

Функция упрощается

Строится схема, реализующая данную функцию

При решении практических задач синтеза одной из существенных является задача минимизации булевых функций.

Минимизация булевых функций осуществляется в два этапа:

На первом этапе находятся все простые импликанты булевой функции.

На втором этапе находятся приведенные системы простых импликант, строятся тупиковые ДНФ, из числа которых отбираются минимальные.

**6. Синтез комбинационных схем с использованием свойств не полностью определенных функций.**

**7. Синтез комбинационных схем в базисах Шеффера и Пирса.**

**8. Языки проектирования цифровой аппаратуры.**

**9. Примеры описания одноразрядного двоичного сумматора на языке Verilog.**

**10. Уровень микроархитектуры (микропрограммный).**

Уровень структурной организации(микроархитектуры) К структурной организации ЭВМ относится физическая модель, которая определяет состав и порядок взаимодействия цифровых логических схем и устройств вычислительной системы. Структурная организация ЭВМ предполагает детальное рассмотрение операционных устройств, которые обеспечивают реализацию заданных архитектурой спецификаций. На данном этапе детально описывается процесс выполнения машинных команд, осуществляемый центральной частью ЭВМ. Операционные устройства центрального процессора работают под управлением сигналов, вырабатываемых устройствами управления (управляющими автоматами, УА). Управляющие сигналы, которые вырабатывают УА (с жёсткой или программируемой логикой), инициируют в операционном автомате (ОА).

**11. Логические схемы алгоритмов. Граф-схемы алгоритмов. Содержательная и закодированная ГСА.**

Для описания строения алгоритмов служит специальный математический аппарат - логические схемы алгоритмов.

Логической схемой алгоритма (ЛСА) называется выражение, состоящее из операторов и логических условий, следующих друг за другом, и нумерованных стрелок, расставленных определенным образом.

Операторы и логические условия называются членами данной ЛСА, стрелки членами ЛСА не являются.

Если алгоритм не является строго последовательным, то в ЛСА включаются логические условия, определяющие порядок выполнения операторов. От каждого логического условия начинается нумерованная стрелка (­i), которая оканчивается у какого-либо члена ЛСА (¯i).

Знак ­i означает начало стрелки, знак ¯i - ее конец. Это как бы одна стрелка `é``¯ у которой опущена средняя часть, чтобы не затемнять ЛСА. Начало и конец одной и той же стрелки обозначаются одинаковыми номерами: ­1 и ¯1 или ­2 и ¯2 и т.д.

В логической схеме алгоритма не может быть двух одинаковых (с одинаковыми индексами) концов стрелок, тогда как одинаковых начал может быть несколько.

определенную последовательность элементарных действий (микроопераций), составляющую микропрограмму операции.

Граф-схема алгоритмов (ГСА)

При задании микропрограммных устройств с помощью графов с каждой вершиной графа сопоставляется микрокоманда (оператор или логические условие алгоритма). Дугам, соединяющим вершины, придается смысл указателя последовательности микрокоманд (выполнения операторов и логических условий).

Граф-схема алгоритма (ГСА) удовлетворяет следующим условиям:

= содержит конечное число вершин, каждая из которых является либо начальной, либо конечной, либо операторной, либо условной;

= имеет одну начальную и одну конечную вершины;

= входы и выходы вершин соединяются друг с другом с помощью дуг, направленных всегда от выхода ко входу;

= каждый выход соединен только с одним входом;

= любой вход соединяется хотя бы с одним выходом;

= для любой вершины графа существует хотя бы один путь из этой вершины к конечной вершине;

= один из выходов условной вершины может соединяться с ее входом (рисунок 4.5 г), что недопустимо для операторной вершины (так называемая ждущая вершина);

= в каждой вершине ГСА записывается оператор, присвоенный вершине.

Если в вершинах ГСА операторы записаны в содержательных терминах, т.е. отражают содержание операций (микрокоманд), такую ГСА называют содержательной граф-схемой алгоритма.

В закодированной ГСА, микрооперации и условия заменены идентификаторами

**12. Язык функционального микропрограммирования. Классы микроопераций.**

Для записи функциональных микропрограмм используется язык функционального микропрограммирования (Ф - язык).

Если средства языка ориентированы на запись микропрограммы без привязки к конкретным структурам для реализации этой микропрограммы, то такой ЯМП называют языком функционального микропрограммирования, а соответствующие микропрограммы — функциональными микропрограммами.

Объектами, описываемыми средством Ф - языка, являются функциональные микропрограммы, определяющие алгоритм выполнения операций в устройствах. Функциональная микропрограмма содержит в себе две части:

1. Описание слов и массивов, устанавливающее типы и форматы слов, с которыми оперирует микропрограмма.

2. Содержательный граф микропрограммы, который определяет алгоритм выполнения операций в содержательной форме – в виде описания микроопераций и логических условий.

Основным элементом информации, с которым оперирует Ф – микропрограмма, является слово. Наименование и формат слова задаются в следующем виде C(П1:П2), где С – идентификатор; П1 и П2 – номера старшего и младшего двоичных разрядов слова. Разряды слова номеруются направо неотрицательными целыми числами n1>0, n2>0, n1<n2. Одноразрядное слово состоит только из идентификатора С.

Совокупность слов, имеющих одинаковую длину, может объединятся в массив. Массив описывается в следующем виде: M[m1:m2](n1:n2), где M – идентификатор массива; m1,m2 – границы номеров слов, составляющих массивов, причем m1<m2; n1,n2 – номера старшего и младшего разрядов слова.

В зависимости от способа использования значений выделяют следующие типы слов:

1. Входные слова – значения присваиваются вне микропрограммы и используются внутри микропрограммы.

2. Внутренние – значения присваиваются и используются только внутри микропрограммы.

3. Вспомогательные – значения присваиваются и используются только внутри микропрограммы, но существуют не постоянно, а только в течении такта автоматного времени (типа А).

4. Выходные – значения присваиваются в микропрограмме и используются вне ее (типа О).

Двоичное выражение определяет правило вычисления двоичного значения путем выполнения операций над первичными двоичными значениями. Константы записываются в следующем виде 11002, 148, 1210, С16.

Первичные двоичные значения, слова и поля – объединяются в двоичное выражение с помощью двоичных операций, к которым относятся обычно следующие операции: ù - инверсия, × конкатенация, Ù - конъюнкция, Ú-дизъюнкция, + - сложение, - - вычитание, Å - сложение по модулю 2.

Все микрооперации подразделяются на семь классов:

1. установки;

2. инвертирования;

3. передачи;

4. сдвига;

5. счета;

6. сложения;

7. бинарные логические;

8. комбинированные.

Микрооперация передачи – обеспечивает присваивание слову значение другого слова (А:=В). Микрооперация сдвига служит для изменения положения разрядов слова по отношению к начальному пути перемещения каждого разряда на k позиций влево или вправо. Операции сдвига обозначаются следующим образом:

Rh(A) – удаление из двоичного выражения А k младших правых разрядов, то есть сдвиг на k разрядов вправо.

Lk(A) – удаление из двоичного выражения A k старших левых разрядов, то есть сдвиг на k разрядов влево.

Бинарные логические микрооперации присваивают слову значение, получаемое поразрядным применением микроопераций Ù, Ú, Å к парам соответствующих разрядов слагаемых. Например, С:=АÚВ.

Комбинированные микрооперации – это микрооперации, не принадлежащие ни к одному из вышеперечисленных классов. Комбинированная микрооперация содержит в себе несколько действий, присущих микрооперациям разных классов.

**13. Функциональная микропрограмма. Функциональная и структурная совместимость микроопераций.**

Чтобы синтезировать схему операционного устройства, необходимо принять некоторый способ выполнения операции в устройстве и описать его в строке микропрограммы. Микропрограмма, представляющая функции операционного устройства безотносительно к средствам, которые могут быть использованы для реализации заданной функции, называется функциональной микропрограммой. Функциональная микропрограмма фиксирует в себе алгоритм выполнения операции, рекомендуемый проектировщиком, и используется как исходная форма представления функций устройства, на основе которой синтезируется структура, достаточная для реализации заданных функций.

Совместимость микроопераций

Совместимостью называется свойство совокупности микроопераций, гарантирующее возможность их параллельного выполнения. Различают функциональную и структурную совместимости. Пусть S1, S2, S3, S4 — подмножества слов из множества S. Тогда микрооперации S1:= φ1(S2) и S3:= φ2(S4) называются функционально совместимыми, если S1 ∩ S2 = Ш, то есть если микрооперации присваивают значения разным словам. В функциональных микропрограммах, описывающих алгоритмы выполнения операций без учета структуры вычислительной машины, одновременно могут выполняться только функционально совместимые микрооперации.

Структура ВМ может внести дополнительные ограничения на возможность параллельного выполнения микроопераций. Микрооперации называются структурно несовместимыми, если из-за ограничений, обусловленных структурой ВМ, они не могут выполняться параллельно. Обычно структурная несовместимость связана с использованием микрооперациями одного и того же оборудования.

**14. Уровень архитектуры системы команд.**

Уровень архитектуры системы команд (Instruction Set Architecture, ISA) Был введен, чтобы скрыть от программистов детали структурной организации ЭВМ. Программистов не интересуют детали технической реализации, им важен только набор функций и сервисов, с помощью которых они смогут эффективно решать свои задачи. Поэтому на уровне архитектуры определяется только функциональная организация ЭВМ, то есть абстрактная модель совокупности функциональных возможностей и сервисов, которые будут использоваться для решения задач программирования.

Архитектура компьютера характеризуется набором машинных команд, размерами и форматами регистров для представления данных и управления вычислительными ресурсами. Машинные команды уровня архитектуры реализуются на нижележащем структурном уровке (микроархитектуры).

**15. Структура операционного устройства. Управляющие и операционные автоматы.**

Операционный автомат (ОА) служит для хранения слов информации, выполнения набора микроопераций и вычисления значений логических условий, т.е. операционный автомат является структурой, организованной для выполнения действий над информацией. Микрооперации, выполняемые ОА, задаются множеством управляющих сигналов Y={y1,....,yM}, с каждым из которых отождествляется определенная микрооперация.

Значения логических условий, вычисляемые в операционном автомате, отображаются множеством осведомительных сигналов X={x1,...,xL}, каждый из которых отождествляется с определенным логическим условием.

Управляющий автомат (УА) генерирует последовательность управляющих сигналов, предписанную микропрограммой и соответствующую значениям логическим условий. Иначе говоря, управляющий автомат задает порядок выполнения действий в ОА, вытекающий из алгоритма выполнения операций. Наименование операции, которую необходимо выполнить в устройстве, определяется кодом g операции, поступающим в УА извне. По отношению к УА сигналы g1,...,gh, посредством которых кодируется наименование операции и осведомительные сигналы x1,...,xL, формируемые в операционном автомате, играют одинаковую роль: они влияют на порядок выработки управляющих сигналов Y. Поэтому сигналы g1,...,gh и x1,...,xL относятся к одному классу – к классу осведомительных сигналов, поступающих на вход УА.

Т.о. любое операционное устройство – процессор, канал ввода-вывода и т.д. – является композицией операционного и управляющего автоматов. Операционный автомат, реализуя действия над словами информации, является исполнительной частью устройства, работой которого управляет управляющий автомат, генерирующий необходимые последовательности управляющих сигналов.

Операционный и управляющий автоматы могут быть определены своими функциями – перечнем выполняемых ими действий.

**16. Структурная организация операционных автоматов.**

Функции ОА сводятся к вводу-выводу и хранению слов информации, выполнению микроопераций и вычислению логических условий. Чтобы реализовать эти действия, необходим набор элементов, достаточный для построения структур с заданными функциями. Такой набор элементов называется структурным базисом ОА.

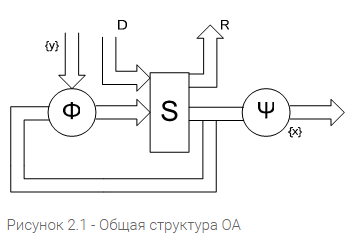
Основными операционными элементами, используемыми в ОА, являются:

управляемые шины, обеспечивающие передачу информации;

регистры;

комбинационные схемы, используемые для выполнения преобразований и формирования логических условий.

В общем виде структура ОА представлена на рисунке 2.1, где Ф - комбинационные схемы, выполняющие преобразование, S - память, ? - комбинационные схемы, формирующие осведомительные сигналы.



**17. Представление чисел в разрядной сетке операционного автомата. Погрешности представления чисел.**

Формой представления чисел в цифровых автоматах называется совокупность правил, позволяющих установить взаимное соответствие между записью числа и его количественным эквивалентом.

Машинное (автоматное) изображение числа это есть представление числа в разрядной сетке цифрового автомата.

Представление числовой информации в ЭВМ, как правило, влечет за собой появление погрешностей (ошибок), величина которых зависит от формы представления чисел и от длины разрядной сетки цифрового автомат.

**18. Сравнительный анализ типов разрядных сеток.**

В микроЭВМ применяют две формы представления чисел: для целых чисел с фиксированной запятой и для представления чисел с плава­ющей запятой (экспоненциальная форма). Форма обычно используе­мых данных называют разрядной сеткой. Представление чисел в разрядной сетке с фиксированной точкой масштабируется, а разрядная сетка с плавающей точкой в масштабировании не нуждается.

**19. Расчёт разрядной сетки.**

В микроЭВМ применяют две формы представления чисел: для целых чисел и для представления чисел с плава­ющей запятой (экспоненциальная форма). Форма обычно используе­мых данных называют разрядной сеткой микроЭВМ.

Можно получить два вида разрядных, сеток для чисел с фиксированной запятой: запятая фиксируется пос­ле младшего разряда (все числа |N|>1); запятая фиксируется пе­ред старшим разрядом (все числа |N|<1).

Для кодирования знака используется "знаковый" (старший) разряд разряд­ной сетки: 0 - соответствует плюсу, а 1 - минусу. При использова­нии обоих видов разрядных сеток для чисел с фиксированной запя­той необходимо, чтобы все данные не выходили за диапазон чисел, допустимых разрядной сеткой. С этой целью используется масштаби­рование.

Представление числа с плавающей запятой не требует масштаби­рования. Общий вид двоичных чисел:  , где q - мантисса; p - порядок в двоичной системе счисления.

**20. Кодирование чисел в прямом, обратном, дополнительном и смещенном кодах.**

Прямой, обратный и дополнительный коды двоичного числа - способы представления двоичных чисел с фиксированной запятой в компьютерной арифметике, предназначенные для записи отрицательных и неотрицательных чисел.

Прямой код - способ представления двоичных чисел с фиксированной запятой. Главным образом используется для записи неотрицательных чисел. В этом варианте (для восьмибитного двоичного числа) мы можем записать максимальное число 255 (всего чисел 256 - от 0 до 255).

Обратный код - метод вычислительной математики, позволяющий вычесть одно число из другого, используя только операцию сложения.

Для неотрицательных чисел обратный код двоичного числа имеет тот же вид, что и запись неотрицательного числа в прямом коде.

Для отрицательных чисел обратный код получается из неотрицательного числа в прямом коде, путем инвертирования всех битов (1 меняем на 0, а 0 меняем на 1).

Дополнительный код - наиболее распространенный способ представления отрицательных чисел. Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел.

Дополнительный код отрицательного числа можно получить двумя способами.

1-й способ:

инвертируем значение отрицательного числа, записанного в прямом коде (знаковый бит не трогаем)

к полученной инверсии прибавляем 1.

2-й способ:

Вычитание числа из нуля

Дано десятичное число 10, необходимо получить отрицательное число (-10) в дополнительном двоичном коде

Переводим 10 в двоичное число:

10 = 0000 1010

Вычитаем из нуля:

0 - 0000 1010 = 1111 0110 - десятичное число -10 в дополнительном коде.

**21. Представление чисел в формате ANSI / IEEE Std754.**

Данный стандарт разработан ассоциацией IEEE (Institute of Electrical and Electronics Engineers) и используется для представления действительных чисел (чисел с плавающей точкой) в двоичном коде. Наиболее используемый стандарт для вычислений с плавающей точкой, используется многими микропроцессорами и логическими устройствами, а также программными средствами.

Основное применение в технике и программирование получили форматы 32 и 64 бита. Чтобы представить число в формате single-precision IEEE 754 необходимо привести его к двоичному нормализованному виду.

Описание преобразования в 32 битный формат IEEE 754: число может быть + или -, поэтому отводится 1 бит для обозначения знака числа: 0-положительное 1-отрицательное Этот самый старший бит в 32 битной последовательности.

Далее пойдут биты экспоненты, для этого выделяют 1 байт (8 бит). Экспонента может быть, как и число, со знаком + или -. Для определения знака экспоненты, чтобы не вводить ещё один бит знака, добавляют смещение к экспоненте в половину байта +127(0111 1111). То есть, если наша экспоната = +7 (+111 в двоичной), то смещенная экспонента = 7+127=134. А если бы, наша экспонента была -7 , то смещенная экспонета=127-7 =120. Смещенную экспоненту записывают в отведенные 8 бит. При этом, когда нам будет нужно получить экспоненту двоичного числа, мы просто отнимем 127 от этого байта.

Оставшиеся 23 бита отводят для мантиссы. Но, у нормализованной двоичной мантиссы первый бит всегда равен 1, так как число лежит в диапазоне 1<=M<2. Нет смыла, записывать единицу в отведенные 23 бита, поэтому в отведенные 23 бита записывают остаток от мантиссы.

**22. Сложение чисел с фиксированной запятой на двоичных сумматорах прямого, обратного и дополнительного кодов.**

При сложении с фиксированной запятой положительные числа остаются в прямом коде, а отрицательные числа переходят в обратный или дополнительный код. Используются правила сложения двоичный арифметики: 0+0=0; 1+0=1; 1+1=10.

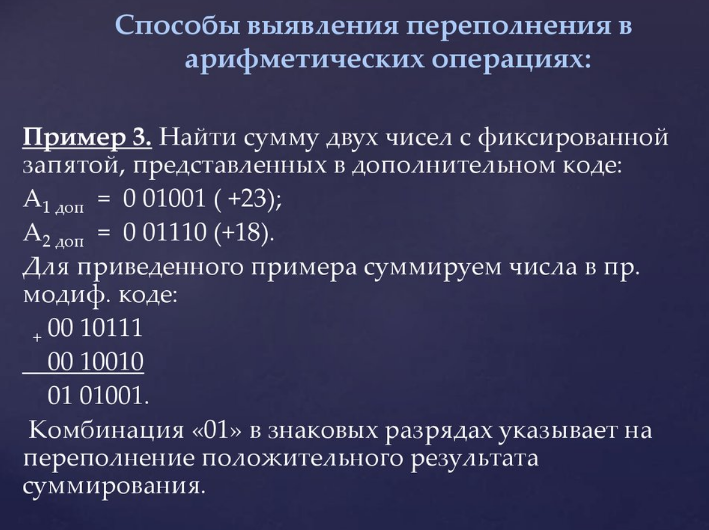
Знаковый разряд участвует в суммировании, так же, как и значащие. При возникновении единицы переноса из знакового разряда для дополнительных кодов, она отбрасывается, а для обратных кодов прибавляется к младшему разряду суммы. Положительный сумма получается в прямом коде, а отрицательные в коде представления слагаемых.

**23. Переполнение разрядной сетки. Динамические и статические методы выявления переполнения.**

В результате выполнения операции сложения может получиться результат, превышающий максимально возможное число для заданной разрядной сетки, т.е. происходит выход полученного результата за пределы разрядной сетки в сторону знакового разряда, называемые переполнением( значащий разряд становится знаковым, результат операции неверный).

При статическом анализе (static analysis) можно обнаружить много разнообразных дефектов и слабых мест исходного кода даже до того, как код будет готов для запуска. С другой стороны, динамический анализ (runtime analysis), или анализ во время выполнения, происходит на работающем программном обеспечении и обнаруживает проблемы по мере их возникновения, обычно используя сложные инструментальные средства. Кто-то может возразить, что одна форма анализа предваряет другую, но разработчики могут комбинировать оба способа для ускорения процессов разработки и тестирования, а также для повышения качества выдаваемого продукта.





**24. Типы сдвигов, используемые в цифровых устройствах.**

Последовательные (сдвигающие) регистры [представляют собою цепочку разрядных схем](https://topuch.ru/resheniya-i-kriterii-ocenivaniya-zaklyuchitelenij-tur-olimpiad/index.html), связанных цепями переноса. Основной режим работы — сдвиг разрядов кода от одного триггера к другому на каждый импульс тактового сигнала.

В однотактных регистрах со сдвигом на один разряд вправо слово сдвигается при поступлении синхросигнала. Вход и выход последовательные @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

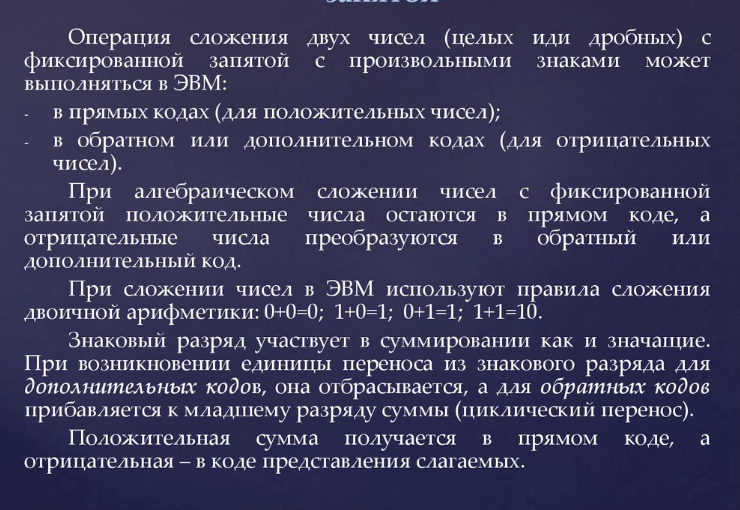
**25. Сложение чисел в формах с фиксированной и плавающей запятой.**

Положительные числа остаются без изменения (в прямом коде), отрицательные переводятся в дополнительный код

Суммируются полученные коды чисел, включая знаковые разряды. Если имеет место перенос из знакового разряда, он отбрасывается.

Анализируется сумма на переполнение. Если имеет место переполнение, то вырабатывается сигнал фи = 1 и ЭВМ останавливает решение задачи

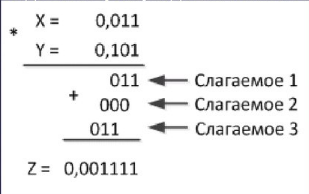
Если переполнения нет, то анализируется результат по знаковому разряду: 0 - результат в прямом коде, 1 - результат в дополнительном коде

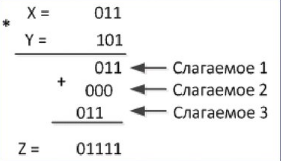




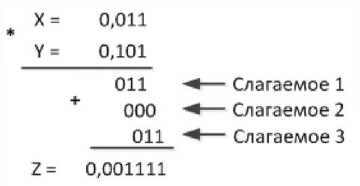
**26. Базовые алгоритмы умножения чисел в форме с фиксированной запятой.**

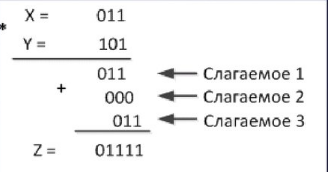
Умножение в столбик начиная с младшего разряда:

а) дробные числа

б) целые числа

Умножение в столбик начиная со старшего разряда:

а) Дробные 

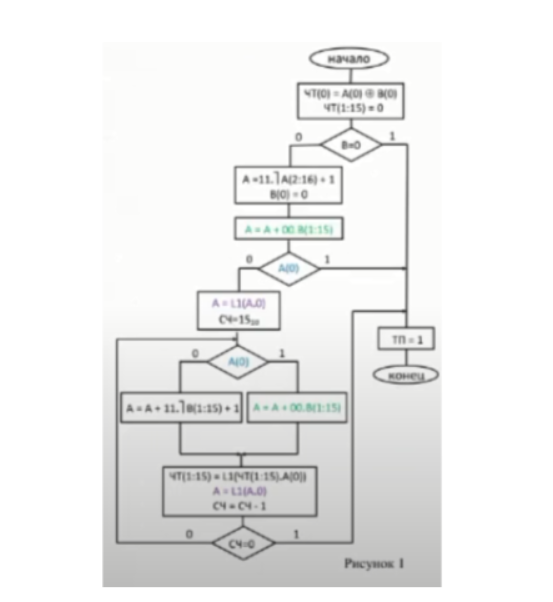
б)Целые 

**27. Сравнительный анализ базовых алгоритмов умножения чисел в форме с фиксированной запятой.**

**28. Умножение чисел в форме с плавающей запятой.**

**29. Деление чисел с фиксированной запятой методом с восстановлением остатка.**

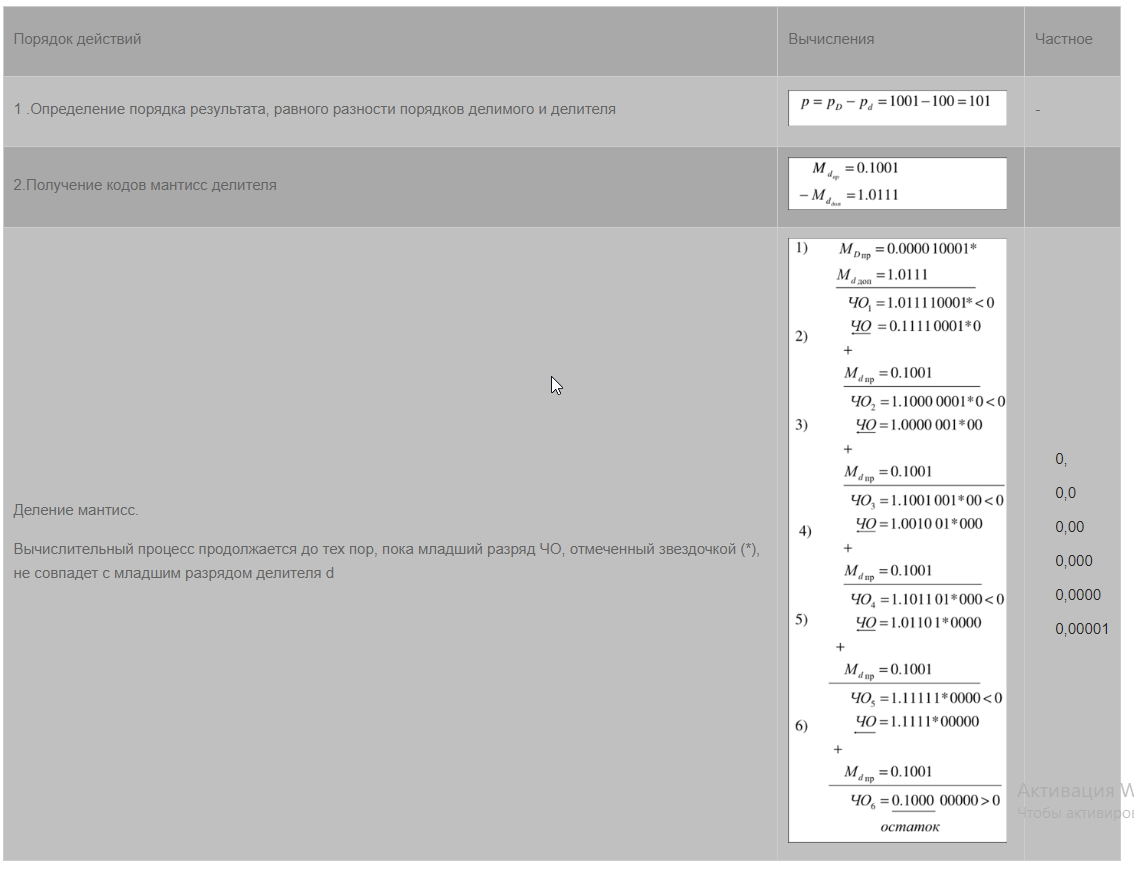
**30. Деление чисел с фиксированной запятой методом без восстановления остатка в прямом коде.**



**31. Деление чисел с фиксированной запятой методом без восстановления остатка в дополнительном коде.**

**32. Деление чисел в форме с плавающей запятой.**

При делении чисел, представленных в форме с плавающей запятой, частное определяется как деление делимого D на делитель d, а порядок - как разность порядков делимого и делителя. Деление мантисс чисел производится по тем же правилам, что и деление чисел с фиксированной запятой.



**33. Извлечение квадратного корня из чисел в формах с фиксированной и плавающей запятой.**

**34. Методы ускорения выполнения арифметических операций.**

Методы ускорения умножения принято делить на аппаратные и логические. Как те, так и другие требуют дополнительных затрат оборудования. При использовании аппаратных методов дополнительные затраты оборудования прямо пропорциональны числу разрядов в операндах. Эти методы вызывают усложнение схемы операционного автомата АЛУ.

Дополнительные затраты оборудования при реализации логических методов ускорения умножения не зависят от разрядности операндов. Усложняется в основном схема управления АЛУ. В ЭВМ для ускорения умножения часто используются комбинации этих методов.

К аппаратным методам ускорения умножения относятся ускорение выполнения операций сложения и сдвига, введение дополнительных цепей сдвига, позволяющих за один такт производить сдвиг информации в регистрах сразу на несколько разрядов, совмещение во времени операций сложения и сдвига, построение комбинационных схем множительных устройств, реализующих "табличное" и "матричное" умножение.

Метод табличного умножения позволяет получить произведение за один такт при условии, что вся таблица умножения (результаты умножения всевозможных пар п-разрядных сомножителей!) будет размещена в памяти. Очевидно, для этого понадобится запоминающее устройство объемом 22п 2n-разрядных слов (точно таким же способом можно выполнять и другие "длинные" операции — деление, вычисление функций). Так, для организации 8-разрядного умножителя потребуется память объемом 216 х16 бит=128 Кбайт, что для современного уровня развития интегральной технологии не кажется чрезмерным.

Матричный метод умножения. Когда множимое и множитель расположены в регистрах машины, нетрудно образовать сразу все частичные произведения. Следовательно, при наличии дополнительных сумматоров можно складывать сразу несколько частичных произведений, а в предельном случае и все. В этом случае формирование произведения можно себе представить как спуск по дереву сумматоров от слагаемых до их общей суммы. Время спуска по дереву будет зависеть от его организации.

Среди логических наиболее распространены в настоящее время методы, позволяющие за один шаг умножения обработать несколько разрядов множителя. Рассмотрим один из способов умножения на два разряда множителя, начиная с его младших разрядов. В зависимости от результата анализа пары разрядов множителя предусматриваются следующие действия (табл. 3.3).

Таблица 3.3. Действия

| Комбинация | Действие | Добавлено |
| --- | --- | --- |
|  | Сдвиг — Сдвиг |  |
|  | Сложение — Сдвиг — Сдвиг | А |
|  | Сдвиг — Сложение — Сдвиг | 2А |
|  | Сложение — Сдвиг — Сложение — Сдвиг | 3А=4А - А |

Таким образом, для умножения сразу на два разряда множителя достаточно:

- при 00 просто произвести сдвиг на два разряда;

- при 01 прибавить к сумме частичных произведений множимое и произвести сдвиг на два разряда;

- при 10 прибавить к сумме частичных произведений удвоенное множимое и произвести сдвиг на два разряда;

- при 11 вычесть из суммы частичных произведений множимое (или добавить обратный (дополнительный) код множимого), произвести сдвиг на два разряда и добавить 1 к следующей (старшей) паре цифр множителя. При классическом методе умножения двоичных п-разрядных чисел согласно выражению (3.26) потребуется n сдвигов суммы частичных произведений и n/2 (в среднем) сложений множимого с суммой частичных произведений. Один из методов ускорения операции умножения — анализ сразу двух разрядов множителя. Это позволит получить результат, применяя n/2 сдвигов и (в среднем) 3n/8 сложений/вычитаний.

**35. Кодирование информации как средство обеспечения контроля работы ЭВМ.**

**36. Контроль арифметических и логических операций.**

**37. Каноническая структура операционного автомата, её свойства.**

Структура операционного автомата (состав и связь) зависит от алгоритмов выполнения машинных команд. Если структура уже задана, то один и тот же алгоритм может быть реализован по разному. Большинство операционных частей процессоров современных ЭВМ строится по некоторым устоявшимся схемам. Эти схемы называются каноническими. Одним из вариантов канонической схемы является универсальная каноническая структура операционного автомата с общими МО. Она достаточно эффективна с позиций технологического процесса, т.к. обладает высокой однотипностью узлов.

В соответствии с канонической структурой, в которой машинная операция описывается в виде набора многодоступных функциональных преобразований данных, можно записать:

C = g(f(A),B)), где g - оператор сдвига, f - оператор бинарных арифметико- -логических операций, h - оператор формирования инверсных кодов (дополнительного, обратного), A,B,C - слова внутри памяти операционного автомата процессора.

Логические условия в этом операционном автомате представляются в виде булевой функции φ(z) от некоторых слов памяти операционного автомата.

Каноническая структура напоминает структуру операционного модуля, для примера BC1, если убрать регистр Q (см. рисунок на практике).

Такое каноническое построение операционного автомата заранее определяет какие МО оказываются совместимыми, т.е. структура автомата определяет состав операционной части микрокоманды. Структура определяет также последовательность выполнения МО операционной части микрокоманды. В большинстве современных процессоров в основе построения операционной части базируется на схеме с общими МО.

**38. Синтез канонической структуры операционного автомата.**

Структура автомата синтезируется следующим образом.

1. Словам, описанных в качестве внутренних ставятся в соответствие регистры с длинами, равными длинам слов.

Sm(5:0) – шестиразрядный регистр, Rg(5:0) – шестиразрядный регистр, рр – триггер.

2. Словам, описанных в качестве входных ставятся в соответствие внешние входы схемы, которые соединяются с регистрами шиной, исходящей из входов.

3. Словам, описанных в качестве выходных ставятся в соответствие внешние выходы схемы, которые соединяются с регистрами шиной, входящей в выходы.

4. Каждой микрооперации yi ÎY, описываемой оператором присвоения (:=) ставится в соответствие комбинационная схема φ iÎ φ (рис. 5.2), входы которой подключаются к регистрам и выходы которой соединяются с управляемой шиной регистра. Управляемая шина помечается сигналом yi, инициирующим микрооперацию – присвоение слову значение некоторой функции. Для выполнения микрооперации передачи не требуется, комбинационная схема, вычисляющая значение некоторой функции. Поэтому структурная реализация микрооперации передачи, обеспечивается управляемой шиной, соединяющей регистр с регистром и отмеченной соответствующим управляющим сигналом. Аналогично микрооперации установки (S:= const) реализуется управляющей шиной, начало которой отмечается константой const и соответствующим управляющим сигналом.

5. Каждому логическому условию xf ставится в соответствие комбинационная схема ψ f Î ψ, входы которой соединяются с регистром, а выходы с отмечаются осведомительным сигналом xf . Если ψ f тривиальная функция, которая, например, показывает содержимое разряда (разрядов) регистра, то логическое условие изображается шиной. Выход шины соответствует осведомительному сигналу.

**39. Эквивалентные микрооперации и обобщенные операторы. Структурная организация I – автоматов.**

Структура операционного автомата I-типа условно может быть разделена на три составные части:

1. регистры памяти;

2. комбинационные вычислители, реализующие функции АЛУ;

3. схемы, формирующие признаки результата.

**40. Синтез I – автоматов.**

Операционный автомат I-типа (укрупненно) синтезируется в соответствии со следующими этапами:

1-й этап.Для достижения максимального параллелизма множество микрокоманд, определяемых операторными вершинами ГСА, разбивается на подмножества {Y1},…,{Yk}, которые реализуют преобразование над словами:

2-й этап.На подмножествах {Yi} выделяются классы эквивалентных операторов для АЛУi–Kij, причем в каждый класс j подмножества включаются микрокоманды эквивалентного функционального назначения, приемником результата у которых является регистр RGi.

3-й этап. Для каждого класса j, содержащего не менее двух эквивалентных операторов, строится обобщенный оператор вида

+

4-й этап. Выполняется построение АЛУi путем суперпозиции всех подсхем, соответствующих обобщенным операторам Di.

**41. Структурная организация M – автоматов.**

Автомат содержит в своей структуре арифметико-логическое устройство, набор регистров, схемы формирования признаков, а также шины внутреннего интерфейса. В общем случае устройства данного типа имеют минимальную аппаратную сложность, однако производительность при этом также оказывается минимальной и равной одной микрокоманде за такт. В отличие от автоматов других типов М-автомат содержит только одну обобщенную схему АЛУ, которая выполняет ряд микроопераций из заданного множества {Ф}.

**42. Синтез M – автоматов.**

Как правило, процедура синтеза М-автоматов сводится к проектированию управляемых источников операндов, построению схемы АЛУ комбинационного типа, проектированию схем коммутации результатов вычислений в регистры ОА.

Этапы синтеза М-автомата:

1-й этап – распределение регистров по шинам.

2-й этап синтеза М-автомата – определение форматов слов на ШД.

3-й этап синтеза – анализ преобразований в АЛУ.

4-й этап синтеза М-автомата – кодирование микрокоманд наборами управляющих сигналов.

5-й этап синтеза – составление подмножеств (или классов) эквивалентных операторов.

6-й этап синтеза – построение обобщенных операторов.

7-й этап синтеза – построение структурных схем для реализации обобщенных операторов.

8-й этап синтеза – формирование списка логических условий.

9-й этап синтеза – построение обобщенной схемы операционного устройства.

**43. Структурная организация IM – автоматов с параллельной комбинационной частью.**

IM - автоматами с параллельной комбинационной частью можно рассматривать, как комбинацию из нескольких М - автоматов, имеющих общую память. Они хорошо приспособлены для реализации микропрограмм, в которых присутствует большое число совместимых микроопераций, и линейные участки микропрограммы не содержат микроопераций, связанных с вычислением одного и того же слова.

**44. Синтез IM – автоматов с параллельной комбинационной частью.**

Синтез данного автомата производится путем разбиения исходного множества микроопераций на L подмножеств и синтез M-автоматов для каждого из этих подмножеств. Разбиение производится таким образом, чтобы функционально-совместимые микрооперации, которые в соответствии с микропрограммой должны выполняться одновременно, находились в различных подмножествах.

**45. Структурная IM – автоматов с последовательной комбинационной частью.**

IM - автоматами называются операционные автоматы, структурная организация которых вносит ограничения на совместимость операций, но в то же время обеспечивает выполнение за такт более чем единой микрооперации. Выделяют IM - автоматы с параллельной комбинационной частью и IM-автоматы с последовательной частью.

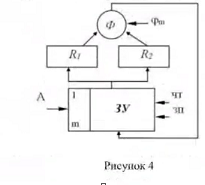
Структура IM-автоматов с последовательной комбинационной частью, как правило, реализуется в виде функционально-законченных устройств с фиксированной системой команд и фиксированной длиной разрядной сетки.

**46. Синтез IM – автоматов с последовательной комбинационной частью.**

Синтез автомата данного типа выполняется на основе выделения из функциональных микропрограмм линейных последовательностей, в каждой из которых выделяются микрооперации, результат выполнения которых присваивается одному и тому же слову. При возможности такие операции объединяются в составные.

**47. Структурная организация S – автоматов.**

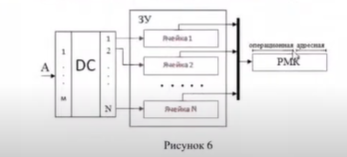
В том случае, когда для выполнения программы необходимо использовать большое число внутренних слов, вместо набора регистров может использоваться оперативное запоминающее устройство. Принцип работы данного класса автоматов, называемых S-автоматов, совпадает с принципом работы М-автоматов.



Этот автомат является самым медленным из рассмотренных, так как запись информации в основную память требует значительно больше времени, чем занесение информации в регистр. Потому для повышения быстродействия S-автоматов результат выполнения команды заносится не в ОЗУ, а на один из регистров R1, R2 и в дальнейшем используется как аргумент в следующей операции. Занесение результата на регистры производится только в том случае, если в следующей операции этот результат используется в качестве аргумента. Это позволяет сократить число обращений к памяти и повысить быстродействие автомата.

**48. Управляющие автоматы с программируемой логикой.**

Идея устройств управления с программируемой логикой основано на том, что микропрограммы можно разместить в запоминающем устройстве (ЗУ). а потом выбирать их из памяти в порядке, определенных схемой алгоритма (рисунок 6). Для инициирования любой микрооперации достаточно сформировать на соответствующей линии управляющий сигнал, для чего в регистр микрокоманд из управляющей памяти микропрограмм последовательно считываются микрокоманды, содержащие информацию о сигналах управления.



Структура управляющего автомата с программируемой логикой слабо зависит от реализуемых алгоритмов, а поведение в основном определяется содержимым ЗУ.  Любое устройство такого типа содержит регистр адреса микрокоманд, запоминающее устройство такого типа содержит регистр адреса микрокоманд, запоминающее устройство, регистр микрокоманд (РМК), схему формирования адреса следующей микропрограммы.

По способу формирования сигналов управления выделяют микропрограммные автоматы с горизонтальным, вертикальным и смешанным кодированием микрокоманд. При горизонтальном кодировании в операционной части под каждый инициирующий микрооперацию сигнал выделяется отдельный разряд.

**49. Горизонтальное, вертикальное и смешанное кодирование микроопераций**

При горизонтальном способе кодирования каждой микрооперации уi ∈ {y1, ..., yn} ставится в соответствие разряд поля микроопераций микрокомандного слова. В этом случае количество разрядов поля микроопераций N равно числу n различных микроопераций, вырабатываемых УА.

Достоинствами горизонтального способа кодирования являются:

- возможность формирования произвольных микрокоманд из заданного набора микроопераций;

- простота реализации схем формирования микроопераций, фактически — их отсутствие, т. к. выход каждого разряда поля микроопераций регистра микрокоманд является выходной линией УА — соответствующей микрооперацией.

Недостаток — чаще всего неэффективно используется память микрокоманд.

При вертикальном способе кодирования в поле микроопераций помещается номер выполняемой микрооперации. При этом количество разрядов N, которое следует предусмотреть в поле микроопераций, определяется выражением: N=k≥log2n.

Достоинство способа в экономном использовании памяти микрокоманд. Недостаток — в невозможности реализовать в микрокоманде более одной микрооперации.

Если реализуемые алгоритмы и структура ОА таковы, что в каждом такте дискретного времени выполняется не более одной микрооперации, то вертикальный способ кодирования — оптимальное решение.

Смешанный способ кодирования, идея которого состоит в следующем. Если во всех микропрограммах, реализуемых УА, нет микрокоманды с большим, чем s, числом микроопераций, то в поле микроопераций можно предусмотреть s подполей разрядностью k, в каждом из которых помещать номер нужной микрооперации. Такой способ позволяет в любой микрокоманде реализовать произвольную s-ку микроопераций, т. е. сохранить гибкость горизонтального кодирования, при возможном значительном сокращении разрядности поля микроопераций: N=s\*k=slog2 n. Эффективность применения смешанного кодирования существенно зависит от значения s, которое может лежать в диапазоне 1≤s≤п. При s=1 имеем случай вертикального кодирования, при s=n — горизонтального.

**50. Распределение микроопераций по полям микрокоманды. Матрицы совместимости и включения.**

Коды микроопераций микрокоманды инициируют совокупность микроопераций, выполняемых за один такт работы. Разряды микрокоманды, отведенные под операционную часть (ОЧ), разбиваются на поля. Каждое поле объединяет некоторый набор несовместных микроопераций, т.е. таких операций, которые не могут выполняться в одном такте. Например, все разные микрооперации, выполняемые на определенном функциональном узле (сумматоре, регистре), являются несовместными.

После определения количества полей и числа микроопераций, включенных в каждое поле, вычисляется длина операционной части микрокоманды по формуле

где n - число полей; mi - число микроопераций, относящихся к n-полю.

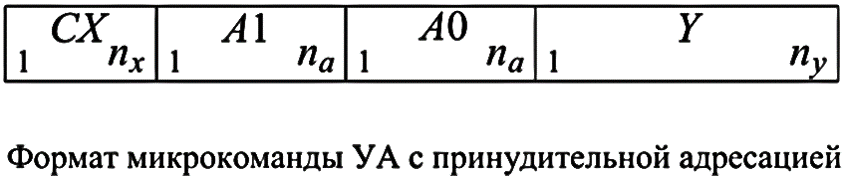
Для вариантов, в которых необходимо синтезировать блок микропрограммного управления микропроцессора, прежде всего следует выписать все различные операторные вершины (без повторений) из составленных микропрограмм выполнения операций в АЛУ и микропрограмм выполнения команд процессором. Этот список и будет представлять собой перечень всех микрокоманд, отличающихся только операционной частью. Данный список вместе с ранее составленными списками микроопераций АЛУ и процессора позволяет построить известными методами матрицу совместимости микроопераций. Ввиду большой размерности такой матрицы заданием предусматривается раздельное построение матриц совместимости микроопераций процессора. (При этом возможное в ряде вариантов использование микроопераций АЛУ для формирования исполнительных адресов не оказывается существенным).

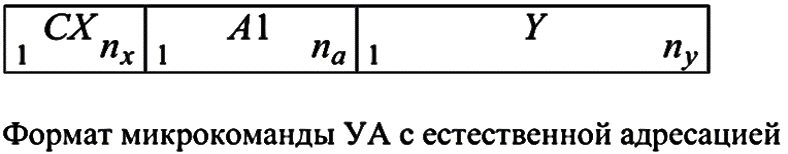
После получения матриц совместимости методом прямого включения строятся подмножества совместимых микроопераций. Допускается построение подмножеств совместимых микроопераций по функциональным признакам, содержательному смыслу микроопераций, однако количество подмножеств при этом может быть слишком большим.

**51. Способы формирования адреса микрокоманд. Принудительная и естественная адресация микрокоманд.**

Способ адресации микрокоманд задаёт правило определения адреса следующей микрокоманды. Существуют два основных способа определения адреса следующей микрокоманды – принудительная адресация и естественная адресация.

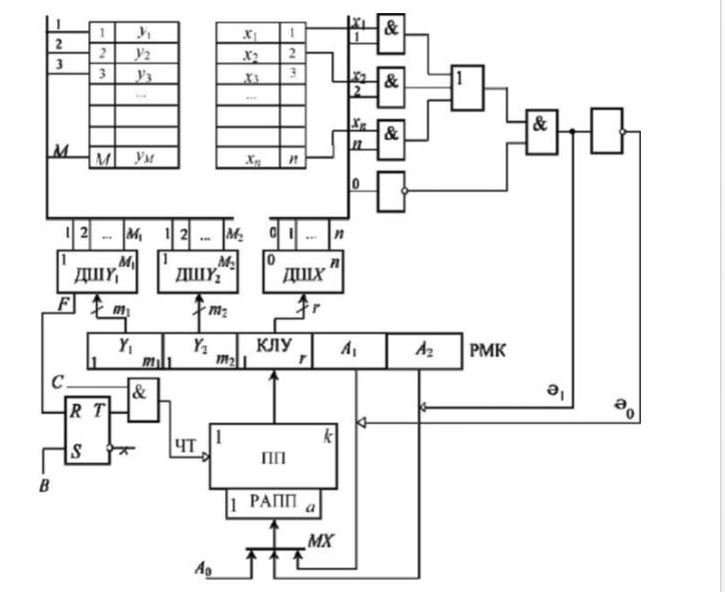
Адресная часть микроинструкции содержит информацию, позволяющую в следующем такте работы выбрать (указать) новый адрес управляющей памяти. Реализация именно этого момента является основным предметом дальнейшего рассмотрения и определяет, в основном, структуру, объем аппаратуры и быстродействие УА.



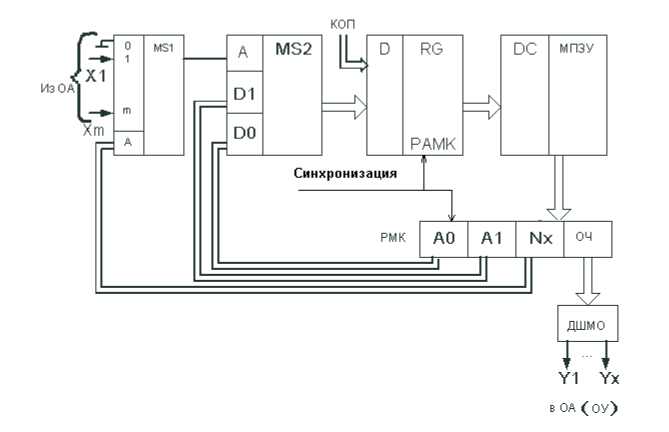


При принудительной адресации адрес перехода к следующему двоичному слову в явном (или неявном) виде присутствует в самой микрокоманде.

**52. Структура управляющего автомата с принудительной адресацией (с двумя адресными полями).**

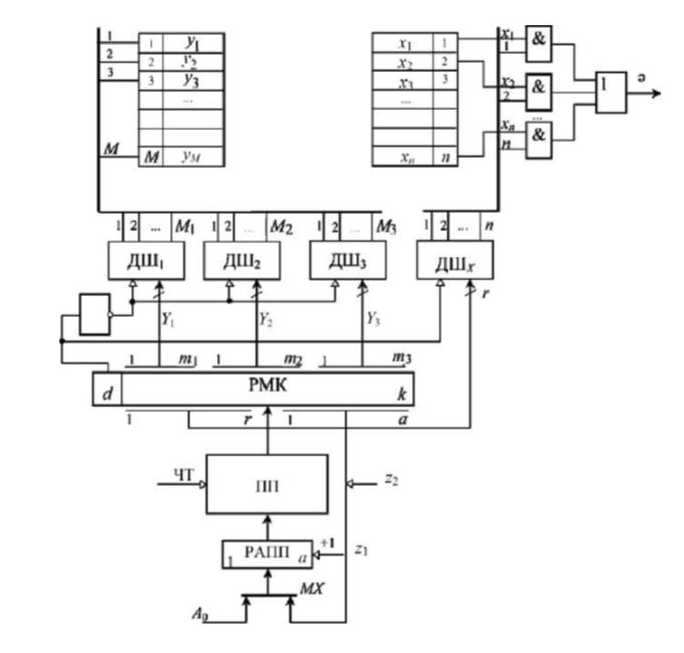


**53. Структура управляющего автомата с принудительной адресацией (с одним адресным полем).**



**54. Синтез управляющих автоматов с принудительной адресацией.**

**55. Структура управляющего автомата с естественной адресацией.**



**56. Синтез управляющего автомата с естественной адресацией.**

**57. Сегментация памяти. Структура управляющего автомата с естественной адресацией и сегментацией памяти.**

Адрес делится на две части:

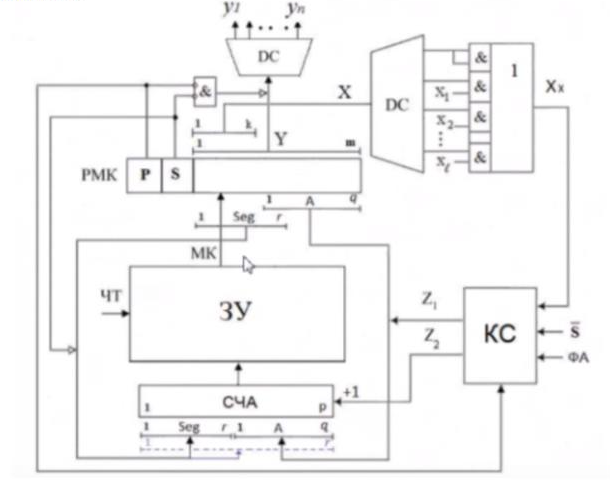
- номер сегмента (1-г)

- относительный адрес ячейки внутри сегмента (1-q)



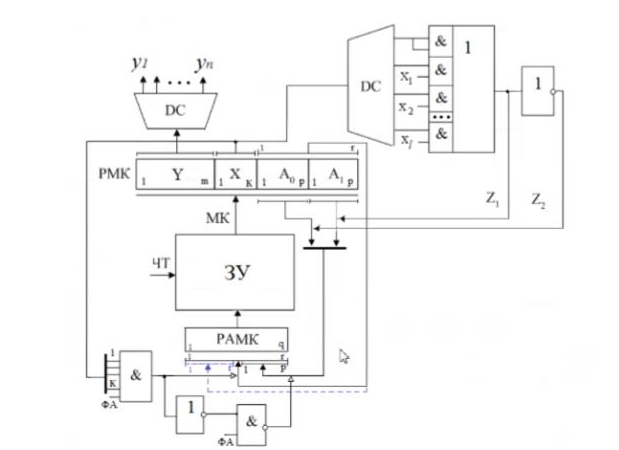
Как правило, в микрокоманде для задания адреса указывается смещение. Для изменения сегмента используется специальная микрокоманда. Для указания того, что данная микрокоманда используется для изменения номера сегмента, вводится дополнительный разряд признака S, либо соответствующим образом кодируется информация в полях Х и Y.

Структура УА (управляющий автомат) с естественной адресацией и сегментацией ЗУ



**58. Структура управляющего автомата с принудительной адресацией и сегментацией памяти.**

Структура УА с принудительной адресацией и сегментацией ПЗУ



**59. Синтез управляющего автомата с естественной адресацией и сегментацией памяти.**

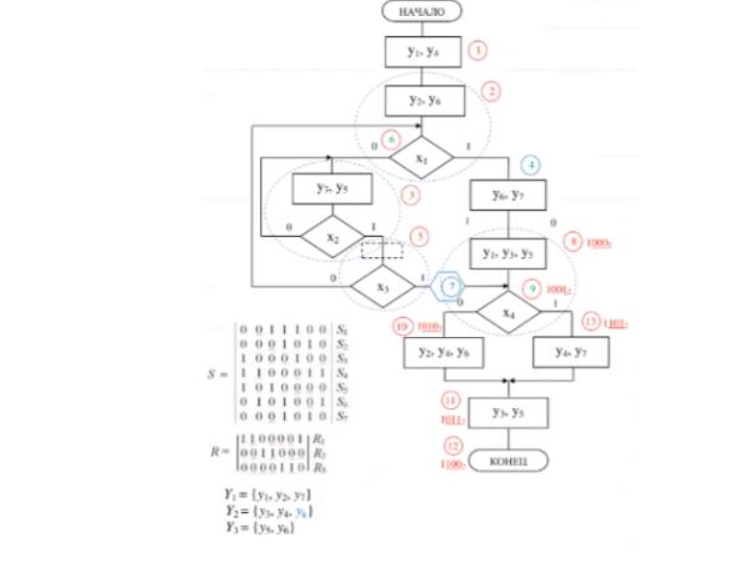
**60. Синтез управляющего автомата с принудительной адресацией и сегментацией памяти.**

Пусть задана закодированная ГСА (рисунок 3).

Память управляющего автомата разбита на секции по 8 ячеек.

Смена сегмента осуществляется, когда в поле X все разряды равны единице.







**61. Структурные методы повышения быстродействия управляющих автоматов с программируемой логикой**

1) Параллельная выборка МК:

В каждой ячейке ЗУ хранятся несколько МК, что позволяет уменьшить количество обращений в IIЗУ.

Адрес очередной МК хранится в СЧАМК и состоит из 2-ух частей:

- адрес участка микропрограммы (1:р),

- номер микрокоманды внутри участка микропрограммы.

При считывании участка микропрограммы, который заносится в РУМП (регистр участка микропрограммы), адрес ячейки ЗУ, по которому производится считывание, заносится в РА. Младшие разряды СЧАМК (P+1, P+2) поступают на дешифратор номера микрокоманды, который осуществляет выборку одной из четырех МК, хранящиеся в РУМП, и передачу ее на выполнение. Если эта МК является операционной, либо управляющей МК, которая не нарушает естественный порядок следования МК, то осуществляется увеличение СЧАМК на 1. Если МК реализует переход по заданному в ней адресу, то адресное поле МК (1: P+2) пересылается в СЧАМК.

Перед выполнением следующей МК производится сравнивание поля (1:P) СЧАМК с содержимым РА. Если сравниваемые коды совпадают, то значит очередная МК находится в РУМП, считывание новой ячейки из ЗУ не производится, а нужная МК определяется разрядами Р+1 и Р+2 СЧАМК. Если сравниваемые коды не совпадают, то производится

считывание нового участка МП из ЗУ (с одновременным занесением адреса этого участка в РА) и выполнение микрокоманды.

2) Опережающая выборка МК:

Процесс выполнения МК можно разбить на 4 временных интервала:

1. время формирования адреса МК - τA

2. время считывания МК из ЗУ - τЗУ

3. время реализации МК - τУ

4. время формирования логических условий - τХ.

3) Однотактная реализация смежных разветвлений.

**62. Управляющие автоматы с жёсткой логикой.**

**63. Математическая модель цифрового автомата (абстрактный автомат).**

Математической моделью цифрового автомата является абстрактный автомат (рисунок 1), который задаётся совокупностью шести объектов:

• конечного множестваZ={z1, z2, ... , zN} входных сигналов (входной алфавит);

• конечного множества W={ω1, ω2, ... , ωG} выходных сигналов (выходной

алфавит);

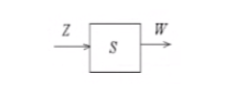
• произвольного множества A={a1, a2, ... , aM} состояний автомата (алфавит

внутренних состояний);

• элемента a1 из множества А, называемого начальным состоянием автомата;

• функцией δ(a,z) переходов автомата;

• функцией λ(a,z) выходов автомата.



**64. Автомат Мили**

В детерминированных (deterministic) автоматах поведение автомата в каждый момент времени однозначно определяется текущей входной информацией и состоянием автомата. В детерминированных автоматах функция переходов δ однозначно определяет состояние aS, в которое переходит автомат из исходного состояния am под действием входного сигнала zf, то есть . Функция выходов λ определяет выходной сигнал, в зависимости от состояния автомата и входного сигнала, то есть .

Выделяют два типа детерминированных автоматов:

1) автомат первого рода, или автомат Мили (George H. Mealy), функционирование которого задаётся уравнениями:

**65. Автомат Мура.**

автомат второго рода, или автомат Мура, заданный уравнениями

Особый интерес на практике представляют автоматы второго рода, у которых выходной сигнал ω(t) определяется одним лишь его состоянием и не зависит явно от входного сигнала z(t), то есть ω(t)=λ(a(t)). Такие автоматы называются автоматами Мура (Edward F. Moore).

**66. Способы задания автоматов. Полностью определенные и частичные автоматы.**

Для задания автоматов используют начальные и автоматные языки описания. К начальным языкам относятся язык регулярных выражений алгебры событий (РВАС), логические схемы алгоритмов (ЛСА), графические схемы (граф-схемы) алгоритмов (ГСА), диаграммы состояний и переходов. Начальные языки описывают автомат на поведенческом уровне, на котором функции переходов и выходов автомата обычно явно не заданы.

При использовании автоматных языков поведение автомата задается путем явного задания функций переходов и выходов. Наиболее часто используются табличный и графический способы задания работы автомата, реже - матрицы переходов и выходов.

Полностью определенным называется абстрактный цифровой автомат, у которого функция переходов и функция выходов определены для всех пар (ai, zj).

Частичным называется абстрактный автомат, у которого функция переходов или функция выходов, или обе эти функции определены не для всех пар (ai, zj).

**67. Преобразование автомата Мура в автомат Мили.**

Пусть задан автомат Мура ([табл. 2.4](https://intuit.ru/studies/courses/1031/242/lecture/6228?page=2#table.2.4)). Таблица переходов эквивалентного автомата Мили ([табл. 2.5](https://intuit.ru/studies/courses/1031/242/lecture/6228?page=2#table.2.5)) совпадает с совмещённой таблицей автомата Мура, представляющей переходы автомата, а [таблица выходов 2.6](https://intuit.ru/studies/courses/1031/242/lecture/6228?page=2#table.2.6) получена следующим образом. Считается, что на переходе из состояния A_m в состояние A_s в эквивалентном автомате Мили должен быть сформирован такой же выходной сигнал, что и в автомате Мура, после того как автомат перешел в состояние а_s, то есть выходной сигнал w_g.

|  | w1 | w2 | w3 | w2 | w3 |
| --- | --- | --- | --- | --- | --- |
|  | a1 | a2 | a3 | a4 | a5 |
| z1 | a2 | a5 | a5 | a3 | a3 |
| z2 | a4 | a2 | a2 | a1 | a1 |

|  | a1 | a2 | a3 | a4 | a5 |
| --- | --- | --- | --- | --- | --- |
| z1 | a2 | a5 | a5 | a3 | a3 |
| z2 | a4 | a2 | a2 | a1 | a1 |

|  | a1 | a2 | a3 | a4 | a5 |
| --- | --- | --- | --- | --- | --- |
| z1 | w2 | w3 | w3 | w3 | w3 |
| z2 | w2 | w2 | w2 | w1 | w1 |

Рассмотрим переход автомата из состояния а1 в состояние а2. В автомате Мура состоянию а2 соответствует выходной сигнал w2, следовательно в [табл.2.6](https://intuit.ru/studies/courses/1031/242/lecture/6228?page=2#table.2.6) на переходе из состояния а1 по входному сигналу z1 ставим w2 и так далее.

При графическом задании автомата Мура переход к автомату Мили выполняется следующим образом: выходной сигнал w_g, формируемый в состоянии A_s, переносится на все дуги, входящие в эту вершину, графическая интерпретация этого показана на [рис.2.4](https://intuit.ru/studies/courses/1031/242/lecture/6228?page=2#image.2.4), а пример трансформации автомата Мура в эквивалентный автомат Мили показан на [рис.2.5](https://intuit.ru/studies/courses/1031/242/lecture/6228?page=2#image.2.5).

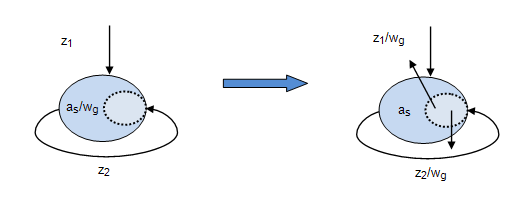
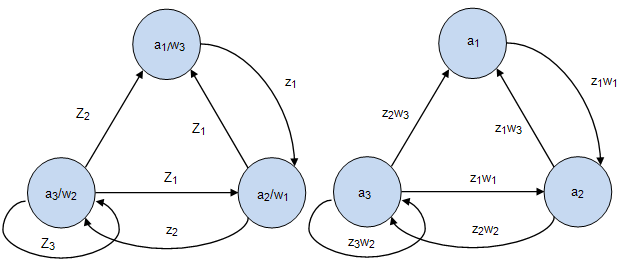


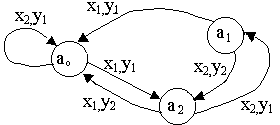
Рис. 2.4.



**68. Преобразование автомата Мили в автомат Мура.**

Ограничение: В автомате Мили не должно быть переходящих состояний, т.е. состояний, в которых имеется хотя бы одна выходящая дуга и не имеется ни одной входящей дуги

Граф автомата Мили



В автомате Мили Xa = {x1, x2}, Ya = {y1,y2}, Aa = {a0, a1,a2}.

В эквивалентном автомате Мура Xb = Xa = {x1, x2}, Yb = Ya = {y1, y2}.

Построим множество состояний Ab автомата Мура, для чего найдем множества пар, порождаемых каждым состоянием автомата Sa.

| Состояние | Порождаемые пары |
| --- | --- |
| a0 | {(a0, y1), (a0, y2)}={b1, b2} |
| a1 | {(a1, y1)}={b3} |
| a2 | {(a2, y1), (a2, y2)}={b4, b5} |

Отсюда имеем множества Ab состояний автомата Мура Ab = {b1, b2, b3, b4, b5}. Для нахождения функции выходов lb с каждым состоянием, представляющим собой пару вида (ai, yg), отождествим выходной сигнал, являющийся вторым элементом этой пары. В результате имеем:

lb(b1) = lb(b3) = lb(b4) = y1;lb(b2) = lb(b5) = y2.

Построим функцию переходов db. Так как в автомате Sa из состояния a0 есть переход под действием сигнала x1 в состояние a2 с выдачей y1,то из множества состояний {b1, b2}, порождаемых a0, в автомате Sb должен быть переход в состояние (a2, y1) = b4 под действием сигнала x1.

Аналогично, из {b1, b2} под действием x2 должен быть переход в (a0, y1) = b1. Из (a1, y1) = b3 под действием x1 переход в (a0, y1) = b1, а под действием x2 – в (a2, y2) = b5. Наконец из состояний {(a2, y1), (a2, y2)} = {b4, b5} под действием x1 в (a0, y2) = b2, а под действием x2 – в (a1, y1) = b3. В результате имеем граф и таблицу переходов эквивалентного автомата Мура.

Граф эквивалентного автомата Мура

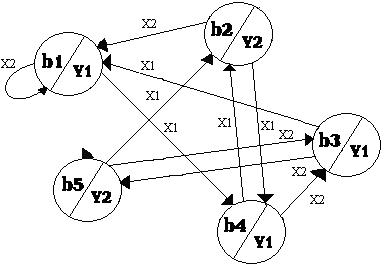


Таблица переходов

| yg | y1 | y2 | y1 | y1 | y2 |
| --- | --- | --- | --- | --- | --- |
| xj\bi | b1 | b2 | b3 | b4 | b5 |
| x1 | b4 | b4 | b1 | b2 | b2 |
| x2 | b1 | b1 | b5 | b3 | b3 |

В качестве начального состояния автомата Sb можно взять любое из состояний b1 или b2, так как оба порождены состоянием a0 автомата Sa.

**69. Минимизация абстрактных таблиц переходов и выходов автомата.**

**70. Абстрактный синтез управляющего автомата Мили (по ГСА).**

**71. Абстрактный синтез управляющего автомата Мура (по ГСА).**

**72. Структурный автомат. Канонический метод структурного синтеза автомата.**

В общем случае задача структурного синтеза автоматов с памятью сводится к нахождению общих приемов построения структурных схем сложных автоматов на основе композиции некоторых элементарных автоматов, т. е. поиску определенных способов их соединения между собой.

Канонический метод структурного синтеза условно можно разделить на следующие этапы:

1) кодирование;

2) выбор элементов памяти автомата;

3) выбор структурно-полной системы элементов;

4) построение уравнений булевых функций выходов и возбуждения автомата;

5) построение функциональной схемы автомата.

**73. Триггеры, основные понятия и классификация.**

Триггер — простейшее последовательностное устройство, которое может находиться в одном из двух возмож­ных состояний и переходить из одного состояния в дру­гое под воздействием входных сигналов.

Триггер является базовым элементом последовательностных логических устройств. Триггеры нашли широкое распространение в вычислительной технике и составляют от 20 до 40% всего оборудования.



Входы триггера имеют различные обозначения, связанные с выполняемыми ими функциями:

S — вход для установки в состояние «1»; S (от англ. set);

R — вход для установки в состояние «О»; R (от англ. reset);

J— вход для установки в состояние «1» в универсаль­ном триггере;

К — вход для установки в состояние «О» в универсаль­ном триггере;

Т— счетный (общий) вход;

D — вход для установки в состояние «1» или в состоя­ние «О»;

V — дополнительный управляющий вход для разреше­ния приема информации (иногда используют букву E вме­сто V).

Выходы триггера обозначают буквами  Q и  , тогда если Q соответствует «1», то  - нулю и наоборот.

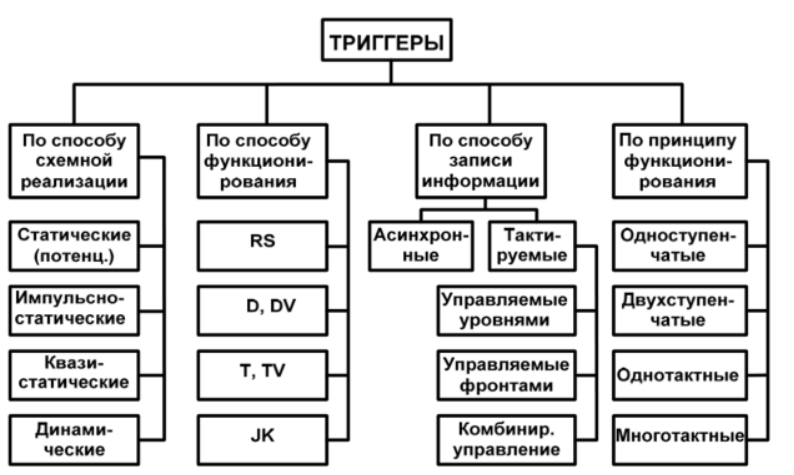
Классификация триггеров может быть осуществлена по различным признакам.

- по способу схемной реализации;

- по способу функционирования;

- по способу записи информации;

- по принципу построения.



В триггерах со статическим (потенциальным) управлением сигнал на входе должен иметь определенную величину (уровень). В триггерах с импульсным управлением срабатывание происходит при подаче на его вход импульса, т.е. сигнала с ограниченной длительностью. Динамические триггеры реагируют на перепад напряжения.

Название триггеров по функциональному признаку совпадает с обозначением его информационных входов.

По способу записи информации триггеры подразделяются на асинхронные и синхронные (тактируемые). Асинхронные триггеры имеют только информационные входы. Запись в такие триггеры информации осуществляется в произвольные моменты времени.

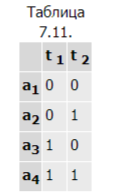
Синхронные триггеры, кроме информационных, имеют дополнительные тактирующие входы С|...СП.Запись информации в такие триггеры осуществляется только в момент действия тактирующего сигнала на синхровходе С.

По принципу построения различают триггеры одноступенчатые и двухступенчатые. В одноступенчатых триггерах поступление сигнала С разрешает их полное переключение в новое состояние.

В двухступенчатых триггерах имеется входной и выходной каскады. Переход триггера в новое состояние происходит в обеих ступенях поочередно. Кроме этого, в двухступенчатых триггерах переход в новое состояние может происходить на одном такте сигнала С или на нескольких. По этой причине различают триггеры однотактные и многотактные.

**74. Синтез автомата Мили с использованием RS–триггеров.**

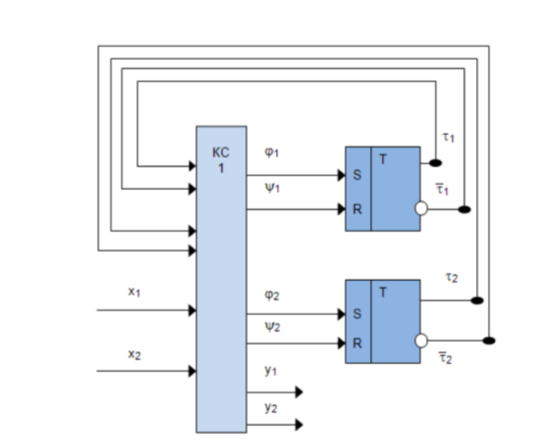
Находим количество элементов памяти R=2 и кодируем состояния абстрактного автомата, например, так, как показано в таблице



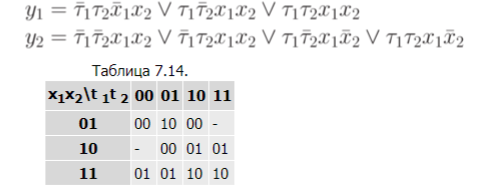
Кодируем входные и выходные сигналы абстрактного автомата, например, так, как показано



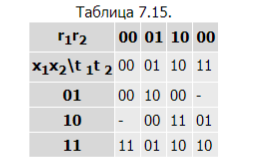
Структурный автомат представляем обобщенной схемой



4. Табл.7.10 представляем, используя коды состояний, входных и выходных сигналов (табл.7.14), и по ней записываем уравнения выходов.

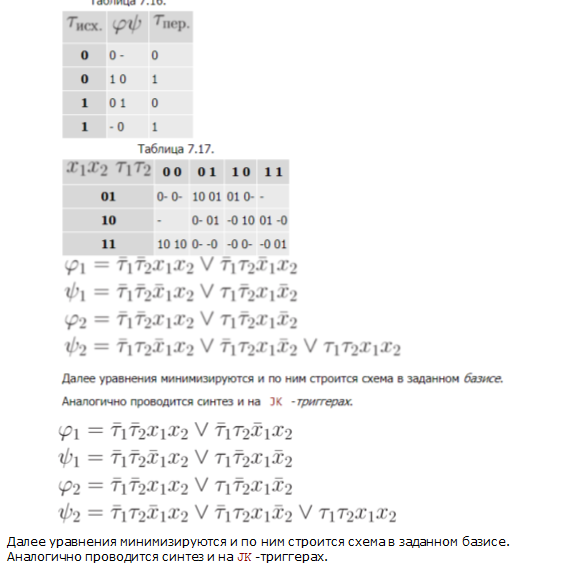


5. Составляем закодированную таблицу переходов автомата (табл.7.15) и по ней записываем уравнения для функций возбуждения.



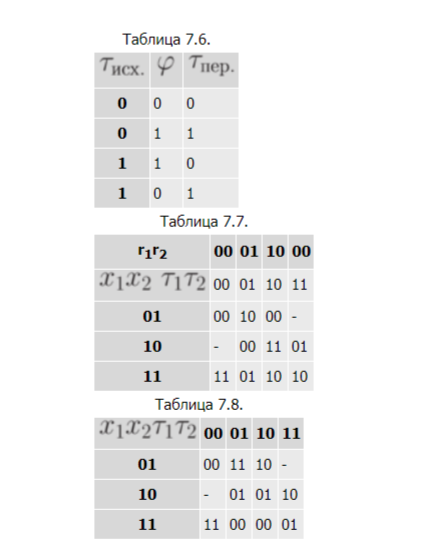
Функция возбуждения RS -триггера представлена в табл.7.16. Просматривая каждый переход триггеров по таблице переходов автомата (табл.7.15), составляем таблицу функций возбуждения (табл.7.17), которая имеет в качестве заголовков столбцов коды состояний, а строки помечены кодами входных сигналов. В каждой клетке таблицы

записаны функции возбуждения для первого триггера и для второго триггера. Составляем для них уравнения:

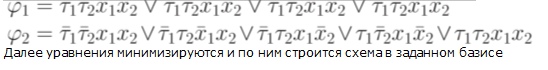


**75. Синтез автомата Мура с использованием D–триггеров.**

Рассмотрим построение уравнений функций возбуждения, то есть начиная с пятого этапа

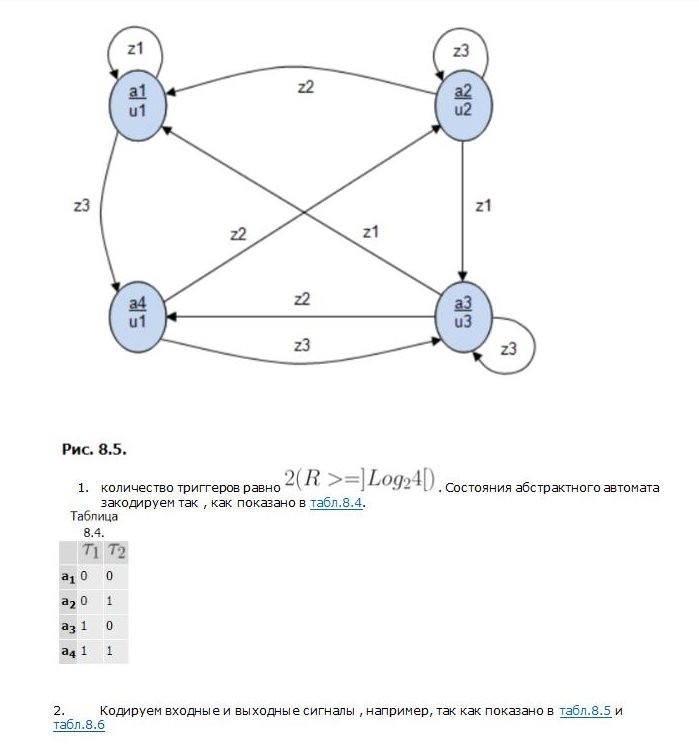


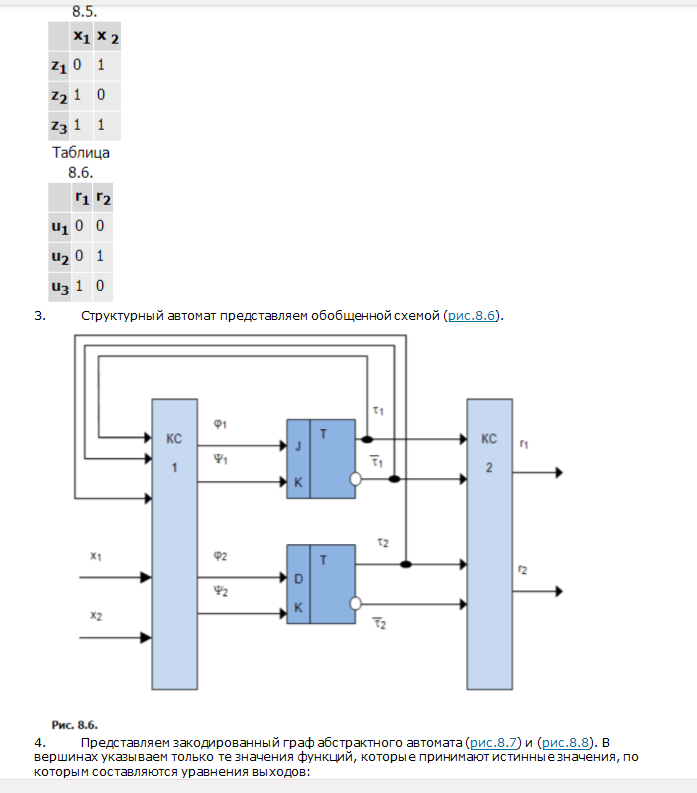
Так как функция возбуждения Т -триггера (табл.7.6) , только тогда, когда состояние автомата переходит из 0 в 1 или из 1 в 0, то по закодированной рис.7.7 переходов исходного автомата Мура находим такие переключения триггеров, при которых они меняли свои состояния. Составляем таблицу функций возбуждения, которая имеет в качестве заголовков столбцов коды состояний, а строки помечены кодами входных сигналов (табл.7.8). В каждой клетке таблицы записаны функции возбуждения. Составляем для них уравнения:

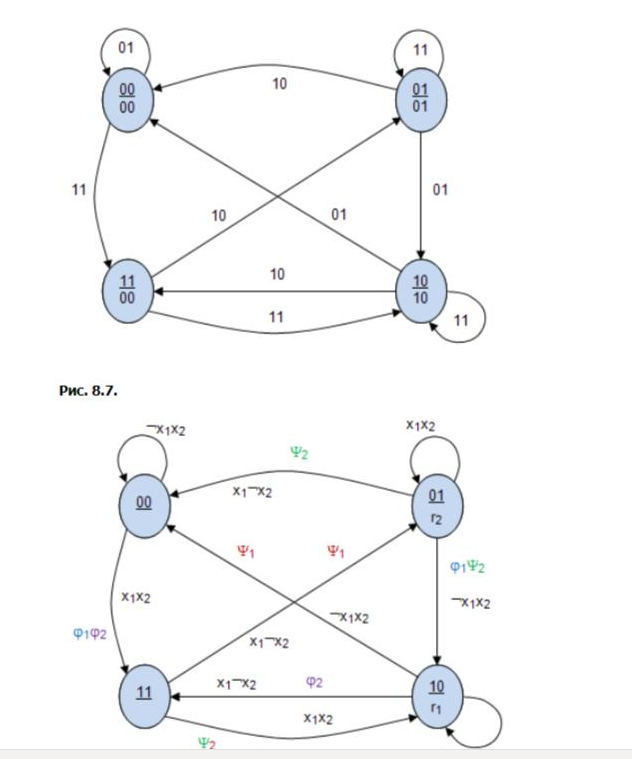


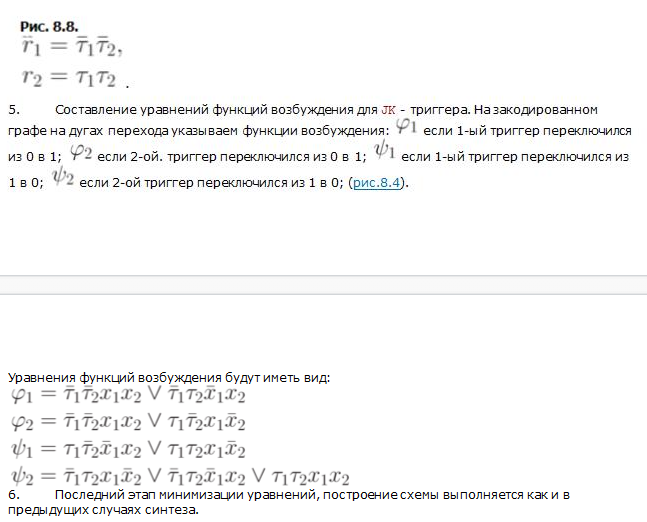
**76. Синтез автомата Мура с использованием JK–триггеров.**

Пусть дан автомат Мура (рис.8.5). Выполним синтез структурного автомата на JK -триггерах.









**77. Синтез автомата Мили с использованием T–триггеров.**

**78. Синтез автоматов в потенциальной системе элементов.**

На асинхронный потенциальный автомат воздействия производят непосредственно его входные информационные сигналы до тех пор, пока сохраняется их активный уровень.

В синхронном автомате используется единый для всего автомата тактовый сигнал, который осуществляет на него импульсное воздействие в соответствии с уровнями входных информационных сигналов, т. е. в синхронном автомате последние только управляют воздействиями, а момент времени воздействия полностью определяется тактовым сигналом. Далее информационные сигналы будут называться просто входными сигналами автомата.

Асинхронный импульсный автомат отличается от потенциального тем, что входные потенциальные сигналы производят на него импульсные воздействия в момент их изменения. Такие воздействия в синхронных автоматах производит только один сигнал — тактовый, а в асинхронных импульсных автоматах любой из входных сигналов может управлять воздействиями и производить их.

**79. Схемотехнические методы обеспечения устойчивой работы автомата (синхронизация элементов памяти, двухступенчатые триггеры, триггеры с динамическим управлением).**

Синхронизация элементов памяти.

В синхронном АП (рис. 28, б) имеются специальные синхросигналы

(тактирующие импульсы) С, которые разрешают элементам памяти прием данных только в определенные моменты времени. Элементами памяти служат синхронные триггеры. Процесс обработки информации упорядочивается во времени, и в течение одного такта возможно распространение процесса переключения только в строго определенных пределах тракта обработки информации.

Двухступенчатые триггеры - триггеры, составленные из двух синхронных одноступенчатых триггеров, причем первая ступень срабатывает с приходом синхросигнала, вторая – по его окончании, при этом состояние первой ступени устанавливается во второй.

Триггеры с динамическим управлением

Запись информации происходит только в момент перехода тактового сигнала L из 0 в 1. При постоянном значении L=0, L=1 или отрицательном перепаде триггер хранит предыдущую информацию, т.е. не обладает свойством прозрачности.

**80. Метод противогоночного кодирования.**

Противогоночное кодирование заключается в развязывании тех пар состояний, для которых осуществляется переход под действием одного и того же сигнала. Пусть (, ) и (, ) - две пары двоичных кодов. Пары (, ) и (, ) называются развязанными, если некоторый разряд кода принимает одно значение на паре (, ) и противоположное - на паре (, ).

Мацевитный Л.В. и Денисенко Е.Л. доказали следующую теорему: в автомате, состояния которого закодированы двоичными кодами конечной длины, гонки отсутствуют тогда, когда для двух любых переходов (am, as) и (ak, al), as  al . происходящих под действием одного и того же входного сигнала, пары кодов состояний развязаны. Авторы приводят алгоритм кодирования.

Алгоритм противогоночного кодирования

1. Выписать все пары переходов, подлежащие развязыванию.

2. Закодировать первую пару кодом 00, вторую - 11.

3. Доопределять следующие пары таким образом, чтобы получались коды 0011 или 1100.

4. Проверить развязку состояний и повторным применением алгоритма добиться минимальной длины кода.

Суть этого алгоритма заключается в том, что, развязывая состояния, мы исключаем критические состязания.

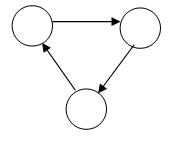
**81. Метод соседнего кодирования.**

Соседнее кодирование состояний автомата

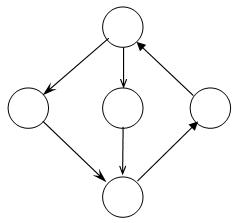
Существует 1 частный способ кодирования - соседнее кодирование. Условие отсутствия гонок при соседнем кодировании всегда выполняется. Суть этого кодирования состоит в том, что 2 состояния связанные дугой графа кодируются наборами, отличающимися лишь в 1 элементе памяти. Существует несколько алгоритмов, но они не всегда поддаются формализации.

Соседнее кодирование однако не всегда возможно. Требование к графу автомата, допускающего соседнее кодирование таково:

1) В графе не должно быть циклов с нечетными числом вершин.



2) 2 соседних состояния не должны иметь более двух состояний, лежащих между ними.



**82. Кодирование состояний автомата и аппаратурные затраты.**

**83. Алгоритм кодирования для D–триггеров.**

Согласно рассматриваемому алгоритму, при кодировании необходимо выполнить следующее:

Каждому состоянию автомата аm (m = 1, 2, ..., M) ставится в соответствие целое число Nm, равное числу переходов в состояние аm (Nm равно числу появлений аm в поле таблицы переходов или числу дуг, входящих в аm при графическом способе задания автомата).

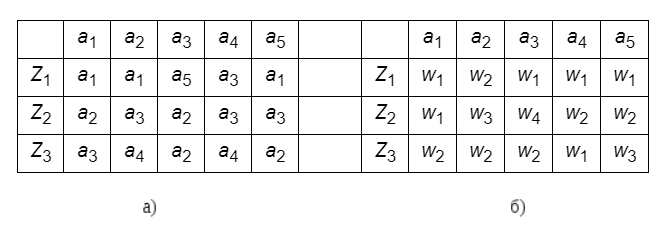
Числа N1, N2, ..., Nm упорядочиваются по убыванию.

Состояние аs с наибольшим Ns кодируется кодом:, где R-количество элементов памяти.

Следующие R состояний согласно списка пункта 2 кодируются кодами, содержащими только одну 1: 00 ... 01, 00 ... 10, ... , 01 ... 00, 10 ... 00.

Для оставшихся состояний опять в порядке списка п.2. используют коды с двумя единицами, затем с тремя и так далее пока не будут закодированы все состояния.

В результате получается такое кодирование, при котором чем больше имеется переходов в некоторое состояние, тем меньше единиц в его коде. Т.к. для D-триггеров функции возбуждения однозначно определяются кодом состояния перехода, то очевидно, что выражения для функций возбуждения будут проще. Этот метод особенно эффективен при отсутствии минимизации функций возбуждения, что имеет место в реальных автоматах с большим количеством внутренних состояний и входных переменных.

В частности, для автомата, заданного своими таблицами переходов и выходов (см. ниже) при кодировании на базе D-триггеров. 

a1 ~ N1 = 3 N3 a3 = 000

a2 ~ N2 = 4 N2 a2 = 001

a3 ~ N3 = 5 N1 a1 = 010

a4 ~ N4 = 5 N4 a4 = 100

a5 ~ N5 = 1 N5 a5 = 011

Аналогично кодированию внутренних состояний для D-триггеров можно кодировать выходные сигналы для любого типа триггеров, т.е. чем чаще вырабатывается данный выходной сигнал wi, тем меньше единиц в его коде. Так для автомата имеем:

w1 ~ N1 = 6 N1 w1 = 00

w2 ~ N2 = 5 N2 w2 = 01

w3 ~ N3 = 2 N3 w3 = 10

w4 ~ N4 = 2 N4 w4 = 11

**84. Эвристический алгоритм кодирования.**

Данный алгоритм минимизирует суммарное число переключений элементов памяти на всех переходах автомата и используется для кодирования состояний автомата при синтезе на базе T, RS, JK-триггеров. Для данных типов триггеров (в отличие от D-триггеров!) на каждом переходе, где триггер меняет свое значение на противоположное, одна из функций возбуждения обязательно равна 1. Уменьшение числа переключений триггеров приводит к уменьшению количества единиц соответствующих функций возбуждения, что при отсутствии минимизации однозначно приводит к упрощению комбинационной схемы автомата.

Пусть Г(S) – неориентированный граф переходов автомата S. Вершины графа отождествляются с состояниями автомата. Вершины i и j соединены ребром, если есть переход из аi и аj или наоборот.

Обозначим q(i, j) число всевозможных переходов автомата из аi в аj. Каждому ребру (i, j) графа Г(S) поставим в соответствие вес ребра р(i, j) = q(i, j) + q(j, i).

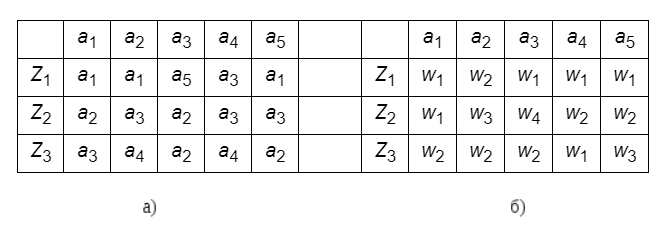
Введем функцию w(i, j) = р(i, j)× d(i, j), где d(i, j) – число компонентов, которыми коды состояний аi в аj отличаются друг от друга (т.е. кодовое расстояние между кодами аi в аj).

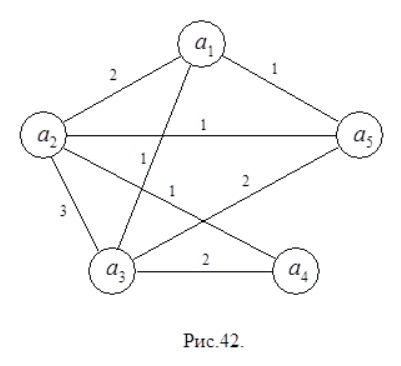
Функция w(i ,j) имеет простой физический смысл. Переход автомата из аi в аj (или наоборот) сопровождается переключением стольких триггеров, сколькими компонентами отличаются коды этих состояний, т.е. их число равно w(i ,j). Следовательно, при переходе автомата по всем ребрам, соединяющим состояниям аi и аj (их число p(i, j)!) всего переключится количество триггеров, равное p(i, j)×d(i ,j) =w(i ,j).

Но тогда функция показывает, сколько всего переключается триггеров при прохождении автомата по всем возможным переходам. Функция w показывает, сколько всего единиц в функции возбуждения, т.е. позволяет оценивать сложность комбинационной схемы автомата. W можно рассматривать как некую целевую функцию, минимум которой определит такое кодирование, при котором комбинационная схема наиболее простая. Кстати, минимальное кодовое расстояние между различными состояниями равно 1 и если удается закодировать все состояния соседним кодированием, то очевидно, что w будет минимально возможным и равным , т.е. суммарному числу переходов для автомата.

Из выражения для w следует, что переход из аi в аi, для которого d(i,i)=0, не влияет на w (что вполне очевидно, если учесть, что на этом переходе ни один триггер не переключается).

Рассмотрим применение эвристического алгоритма на конкретном примере автомата, заданного таблицами переходов и выходов. Для данного автомата можно построить ориентированный граф (без учета петель), представленный на рис.42. На каждом ребре указан его вес.





**85. Традиционная (фон-неймановская) архитектура ЭВМ.**

Принципы фон Неймана

Использование двоичной системы счисления в вычислительных машинах. Преимущество перед десятичной системой счисления заключается в том, что устройства можно делать достаточно простыми, арифметические и логические операции в двоичной системе счисления также выполняются достаточно просто.

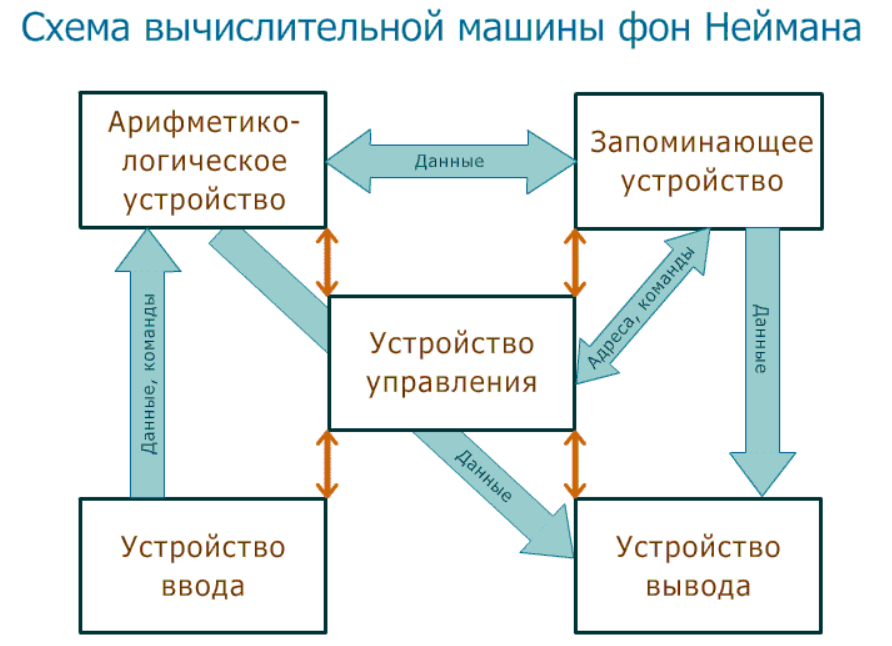
Программное управление ЭВМ. Работа ЭВМ контролируется программой, состоящей из набора команд. Команды выполняются последовательно друг за другом. Созданием машины с хранимой в памяти программой было положено начало тому, что мы сегодня называем программированием.

Память компьютера используется не только для хранения данных, но и программ. При этом и команды программы и данные кодируются в двоичной системе счисления, т.е. их способ записи одинаков. Поэтому в определенных ситуациях над командами можно выполнять те же действия, что и над данными.

Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы. В любой момент можно обратиться к любой ячейке памяти по ее адресу. Этот принцип открыл возможность использовать переменные в программировании.

Возможность условного перехода в процессе выполнения программы. Несмотря на то, что команды выполняются последовательно, в программах можно реализовать возможность перехода к любому участку кода.

Самым главным следствием этих принципов можно назвать то, что теперь программа уже не была постоянной частью машины.



**86. Форматы команд, режимы адресации.**

Форматом команды называется заранее обговоренная структура полей в её кодах, позволяющая ЭВМ распознавать составные части кода.

Под форматом команды следует понимать длину команды, количество, раз­мер, положение, назначение и способ кодировки ее полей.

Команды, как и любая информация в ЭВМ, кодируются двоичными словами, которые должны содержать в себе следующие виды информации:

тип операции, которую следует реализовать в данной команде (КОП);

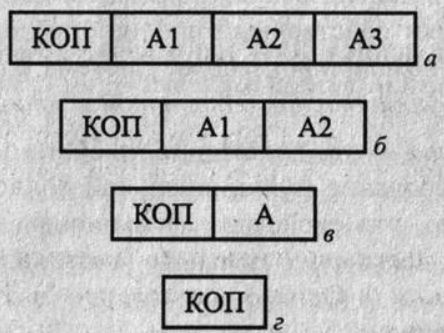
место в памяти, откуда следует взять первый операнд (А1);

место в памяти, откуда следует взять второй операнд (А2);

место в памяти, куда следует поместить результат (A3).

Каждому из этих видов информации соответствует своя часть двоичного сло­ва — поле, а совокупность полей (их длины, расположение в командном слове, способ кодирования информации) называется форматом команды. В свою очередь, некоторые поля команды могут делиться на подполя. Формат команды, поля которого перечислены выше, называется трехадресным.

Форматы команд: а — трехадресный; б — двухадресный; в — одноадресный; г — безадресный



Объем памяти в ЭВМ постоянно растет и поэтому существенной становится проблема сокращения длины команды. Для преодоления ограничений по длине команды и повышения эффективности обработки информации в ЭВМ имеется множество режимов адресации, которые позволяют

определять полный адрес памяти меньшим числом бит, тем самым, сокращая длину команды;

обращаться к ячейкам памяти, адреса которых вычисляются во время выполнения программы, что обеспечивает удобный доступ к данным различной структуры;

вычислять адрес памяти относительно позиции команды или относительно другого объекта таким образом, что программу можно загружать в любую область памяти без всяких изменений адресов в программе.

Основные способы формирования адресов операндов в современных ЭВМ:

Непосредственная адресация. В этом самом простом режиме адресации операнд, подлежащий обработке, размещается непосредственно в самой команде и передается в процессор следом за выборкой кода операции команды. Данный режим адресации применяется тогда, когда операндом является константа, не изменяющаяся во время выполнения программы.

Абсолютная или прямая адресация. Часть команды является адресом операнда в памяти. Этот способ адресации достаточно быстр, так как при поиске адреса не требуется никаких дополнительных вычислений.

Регистровая адресация. В данном режиме операнд находится в одном из регистров процессора и в команде просто указывается номер требуемого регистра. Команда с регистровой адресацией будет достаточно короткой. Кроме того, для выборки операндов процессору не требуется достаточно длительная операция обращения к памяти, поскольку регистры встроены в сам процессор, поэтому команды с регистровой адресацией являются самыми “быстрыми”.

Регистровая косвенная адресация. В этом режиме адресации регистр или пара регистров процессора содержат адрес операнда, и в команде указывается номер регистра с адресом. Адрес может храниться в специализированном указательном регистре или регистре общего назначения. По быстродействию приближается к прямой адресации, так как адрес содержится внутри регистра процессора и для его выборки не требуется производить сравнительно длительную операцию обращения к памяти.

Адресация с автоуменьшением или автоувеличением. При адресации с автоувеличением адрес операнда вычисляется практически так же, как и при косвенной регистровой адресации. После выборки операнда, адрес, находящийся в регистре процессора, автоматически увеличивается на 1 для указания следующего байта, на 2 - для указания поля из двух байт и т.д., при этом размер операнда определяется кодом операции. Отличие адресации с автоуменьшением состоит в том, что перед выборкой операнда содержимое регистра уменьшается на 1, 2 и т.д. в соответствии с размером операнда. Эти два типа адресации позволяют эффективно обрабатывать массивы однотипных данных.

Неявная или подразумеваемая адресация. При таком способе адресации в команде явно не указывается адрес одного или нескольких операндов. Операнды в командах с неявной адресацией могут находиться, например, в выделенных для этой команды регистрах процессора, или некоторый выделенный и заранее известный регистр процессора может содержать адрес операнда.

Индексная адресация удобна для обращения к массивам и таблицам. Для образования исполнительного адреса к адресной части команды прибавляется смещение из индексного регистра, называемое индексом. Индексный регистр является программно - доступным, и его содержимое может изменяться, что позволяет изменять исполнительные адреса без модификации адресной части команды. В качестве индексных регистров используются один или несколько специализированных регистров, в некоторых ЭВМ их функции выполняют регистры общего назначения. Когда индексный режим используется для доступа к массиву, адрес в команде соответствует базовому адресу массива, а значение индексного регистра - индексу компоненты массива.

Относительная адресация. Основное назначение относительной адресации - преодолеть недостатки прямой адресации, сохранив при этом ее внешнюю простоту и эффективность. При относительной адресации в команде указывается не полный адрес операнда, а сокращенный адрес, обычно называемый смещением. В процессе выполнения команды полный адрес операнда вычисляется путем суммирования смещения, указанного в команде с некоторой величиной, которая называется базовым адресом. Существует два способа расширения адреса, указанного в короткой форме: самоопределяющаяся относительная адресация (смещение, указанное в команде, складывается с адресом самой команды) и базовая адресация (смещение, указываемое в команде, складывается с базовым адресом, хранящимся в регистре базы процессора).

Разновидностью базовой адресации является сегментная адресация, применяемая в процессорах семейства 8086.

**87. Управление последовательностью выполнения команд в системах с нетрадиционной архитектурой (управление потоком данных, управление по запросу).**

Способ перехода к выполнению следующего оператора (команды) задается механизмом вызова команд, который определяет порядок выполнения операторов и может быть реализован следующим образом:

команда выполняется, после того как выполнена предшествующая ей команда последовательности (используется принудительный или естественный способ перехода к следующему оператору);

команда выполняется, когда становятся доступными ее операнды (механизм управления данными);

команда выполняется, когда другим командам требуется результат ее выполнения (механизм управления по запросу)

**88. Параллелизм на уровне архитектуры системы команд (Instruction-level Parallelism, IPL).**

Проблема, возникшая в компьютерной отрасли:

Определить новый тип архитектуры, не требующий переписывания программ в «параллельном виде» , смены языка и т. д. , причем обеспечить в нем высокую степень параллелизма, и не слишком усложнять аппаратное обеспечение.

Решение:

Более высокая производительность достигается за счёт:

совершенствования полупроводниковой технологии, т. е. увеличения скорости передачи сигналов в микросхеме.

увеличения плотности микросхем.

реализации определенного вида параллелизма.

Параллелизм на уровне команд (instruction-level parallelism, ILP) - это параллельное выполнение отдельных RISC-операций. Он существенно ускоряет работу процессоров.

Основной плюс: Системы ILP используют программы, написанные на традиционных языках высокого уровня. А обнаружение параллелизма выполняется благодаря компиляторной технологии и апп. обеспечению.

Это решение не требует от программистов дополнительных усилий и предполагает, что можно не переписывать приложения. Сейчас параллельная обработка на уровне команд - единственный надежный подход, позволяющий добиться увеличения производительности без фундаментальной переработки приложений.

**89. Конвейеризация вычислений.**

Конвейеризация - метод проектирования, в результате применения которого в вычислительной системе обеспечивается совмещение различных действий по вычислению базовых функций за счет их разбиения на подфункции.

Условия конвейеризации:

вычисление базовой функции эквивалентно вычислению некоторой последовательности подфункций;

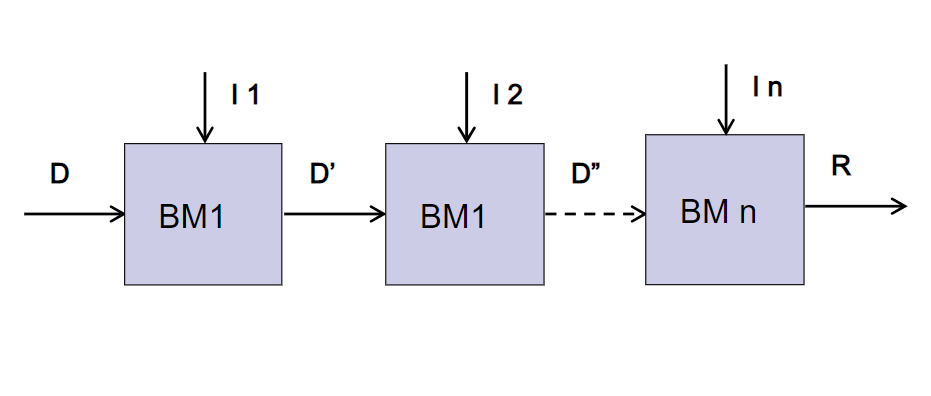
величины, являющиеся входными для данной подфункции, являются выходными величинами той подфункции, которая предшествует данной в процессе вычисления;

никаких других взаимосвязей, кроме обмена данными, между подфункциями нет;

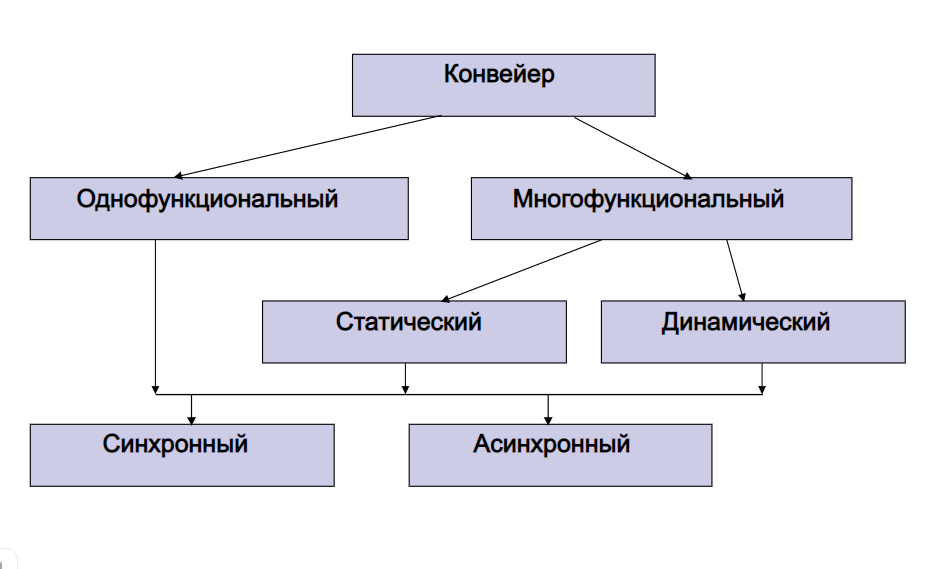
каждая подфункция может быть выполнена аппаратными блоками;

времена, необходимые для реализации аппаратными блоками своих действий, имеют один порядок величины.

Архитектура



Классификация



Задача управления

обеспечение входного потока данных (заполнение конвейера)

задача диспетчеризации - определение моментов времени, в которые каждый элемент входных данных должен начинать свое прохождение по конвейеру.

Проблемы управления

разный период времени обработки данных на разных ступенях;

обратная связь от текущей ступени к какой- либо из предыдущих;

множественные пути от текущей ступени к последующим;

подача элемента данных более чем на одну ступень одновременно (элемент распараллеливания обработки);

существование между входными элементами зависимостей, которые принуждают к определенному упорядочению связанных с ними вычислений;

Стратегия управления

Стратегия управления - процедура, которая выбирает последовательность латентностей.

Жадная стратегия - выбирает всегда минимально возможную латентность между данной и следующей инициацией без учета каких бы то ни было следующих инициаций.

Оптимальная стратегия - обеспечивает минимальную достижимую среднюю латентность.

**90. Матричные системы.**

Матричные системы лучше всего приспособлены для решения задач, характеризующихся параллелизмом независимых объектов или данных. Матричная система состоит из множества процессорных элементов (ПЭ), организованных таким образом, что они исполняют векторные команды, задаваемые общим для всех устройством управления, причем каждый ПЭ работает с отдельным элементом вектора. ПЭ соединены через коммутационное устройство с многомодульной памятью. Исполнение векторной команды включает чтение из памяти элементов векторов, распределение их по процессорам, выполнение заданной операции и засылку результатов обратно в память. Таким образом, производительность системы оказывается равной сумме производительностей всех процессорных элементов.



**91. Систолические системы.**

Систолические системы являются очень специализированными вычислителями и производятся под конкретную задачу. Фактически, задача построения систолического вычислителя сводится к построению аппаратного конвейера, имеющего достаточно большое время получения результата (т.е. большое количество ступеней), но при этом сравнительно маленькое время между последовательной выдачей результатов, так как значительное количество промежуточных значений обрабатывается на разных ступенях конвейера.

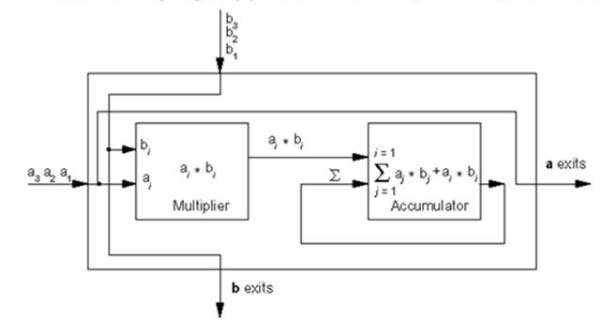
Базовые принципы построения систолических архитектур.

Систола представляет собой сеть связанных вычислительных ячеек, обычно простых;

Каждая ячейка содержит в себе буферный входной регистр для данных и вычислитель, оперирующий с содержимым этого регистра. Выход вычислителя может подаваться на входы других ячеек;

Операции в систоле производятся по типу конвейерной обработки;

Вычисления в систоле регулируются с помощью общего тактового сигнала;



Основные характеристики систолической ВА:

однородность процессорного поля,

регулярность (постоянство) межпроцессорных соединений,

синхронностью функционирования процессорных элементов.

В каждый момент времени выполняются одновременные одинаковые операции или одинаковые вычислительные модули. Такими модулями могут быть:

модули обработки данных и вычислений

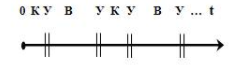
модули, отвечающие за внешнюю коммуникацию.

Каждый из двух типов этих модулей выполняется в свою фазу обработки. Фазы обработки систолических ВА:

К: внешняя коммуникация между ПЭ ( В этот интервал времени во всей процессорной сети одновременно происходит обмен данными между ПЭ. Интервал должен по длительности соответствовать наиболее долгой операции коммуникации в сети.);

В: вычисления в ПЭ (Осуществляет вычисления и обработку информации. Длительность данной фазы должна соответствовать наиболее продолжительному вычислительному модулю.);

У: управление вычислениями и коммуникацией (очень короткая) (Осуществляет выполнение операций по старту и окончанию работ процессорного поля (соответствуют началу и концу каждой вычислительной операции). Останов обработки процессов в любой момент времени до получения результата.).



Применение систолических ВА:

ускорители, встроенные в ПК и реализующие конкретные вычислительные алгоритмы (матричные операции, решение систем линейных алгебраических уравнений, распознавание образов, сортировка и др. ). В этом случае процессорная плата используется в качестве сопроцессора. Время вычислений сокращается на 1 – 3 порядка.

систолические процессоры, встроенные в технические системы, которые используются для цифровой обработки в реальном масштабе времени. Например, алгоритм цифровой фильтрации и др.

**92. Волновые системы.**

**93. Теговые и ассоциативные процессоры.**

Введение тегов, не исключающее применение статических Я ВУ, определило понятие теговой архитектуры. Чтобы отойти от принципа однородности памяти и получить возможность содержательной трактовки хранимых данных предложили информацию, хранящуюся в основной памяти или регистре, снабжать специальным полем (тегом), определяющим тип и формат данных. Теговая организация памяти позволяет достигнуть инвариантности команд относительно типов и форматов данных. Например, при использовании тегов отпадает необходимость различать команды для выполнения арифметических операций с числами в форме с плавающей и фиксированной запятой, так как машина это сделает автоматически, проверив типы операндов. Использование механизмов тегирования дает возможность упростить систему команд процессора, облегчает обнаружение таких ошибок, как несовпадение типов данных, попытка чтения из пустой ячейки, упрощает компиляторы.

Ассоциативный способ обработки данных позволяет преодолеть многие ограничения, присущие адресному доступу к памяти, за счет задания некоторого критерия отбора и проведения необходимых преобразований, только над теми данными, которые удовлетворяют этому критерию. Критерием отбора может быть совпадение с любым элементом данных, достаточным для выделения искомых данных из всех имеющихся. Поиск данных может происходить по фрагменту, имеющему большую или меньшую корреляцию с заданным элементом данных.

Ассоциативные системы относятся к классу SIMD (Single Instruction Multiple Data) - один поток команд, множество потоков данных. Эти системы включают большое число операционных устройств, способных одновременно по командам управляющего устройства вести обработку нескольких потоков данных. В ассоциативных вычислительных системах информация на обработку поступает от ассоциативных запоминающих устройств (АЗУ), характеризующихся тем, что информация в них выбирается не по определенному адресу, а по ее содержанию.



**94. Пример параллельного вычисления арифметического выражения, представленного в форме обратной польской записи.**

Обратная польская запись идеально подходит для вычисления формул на компьютере со стеком. Формула состоит из n символов, каждый из которых является либо операндом, либо оператором. Алгоритм для вычисления формулы в обратной польской записи с использованием стека прост. Нужно просто прочитать обратную польскую запись слева направо. Если встречается операнд, его нужно поместить в стек. Если встречается оператор, нужно выполнить заданную им операцию.

В качестве примера рассмотрим вычисление следующего выражения: (8+2\*5)/(1+3\*2-4). Соответствующая формула в обратной польской записи выглядит так: 825\*+132\*+4-/

Число на вершине стека – это правый операнд (а не левый). Это очень важно для операций деления, вычитания и возведения в степень, поскольку порядок следования операндов в данном случае имеет значение (в отличие от операций сложения и умножения). Другими словами, операция деления действует следующим образом: сначала в стек помещается числитель, потом знаменатель, и тогда операция даёт правильный результат. Отметим, что преобразовать обратную польскую запись в машинный код очень легко: нужно просто двигаться по формуле в обратной польской записи, записывая по одной команде для каждого символа. Если символ является константой или переменной, нужно вписывать команду помещения этой константы или переменной в стек, если символ является оператором, нужно вписывать команду выполнения этой операции.



**95. Параллельные вычислительные системы с общей (Shared) и с распределенной (Distributed) памятью.**

Параллельные вычислительные системы с общей памятью принято называть мультипроцессорами, а с распределённой памятью — мультикомпьютерами. Процессоры в мультипроцессоре взаимодействуют путем обращений к общей памяти, а в мультикомпьютере обмениваются сообщениями через коммуникационную сеть, что относит их к классу более сложных систем, чем мультипроцессоры.

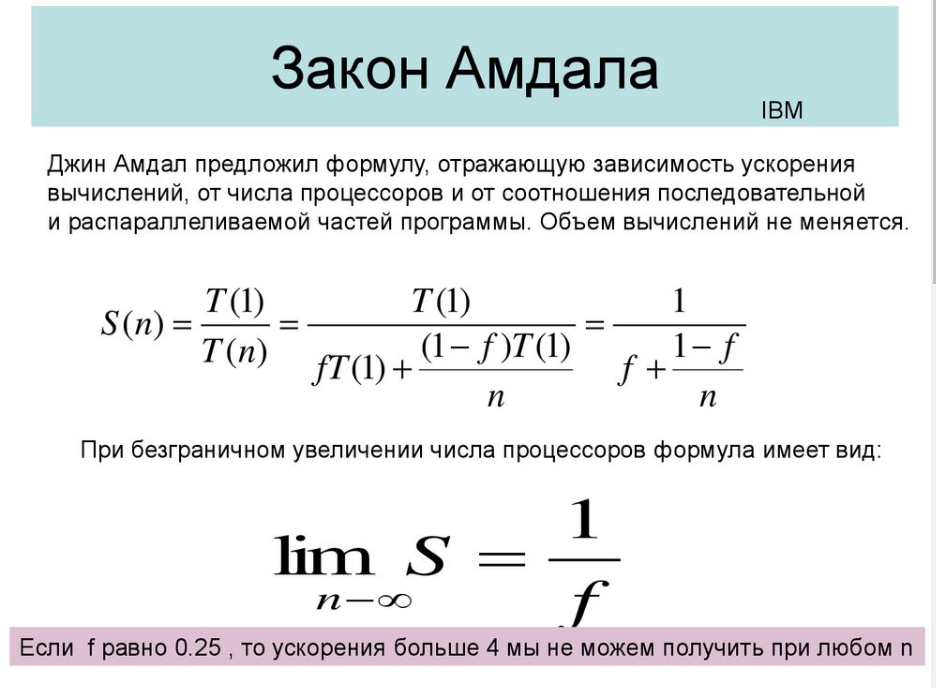
Мультипроцессоры, в особенности мультипроцессоры с симметричной архитектурой, широко используются в настольных офисных системах, однако они плохо масштабируются. Поэтому высокопроизводительные вычисления выполняются на суперкомпьютерах, имеющих распределенную память. Более того, разделение суперкомпьютеров на комплексы с общей памятью и комплексы с распределенной памятью потеряло актуальность, поскольку объединить сотни тысяч ядер в систему с общей памятью технически невозможно.

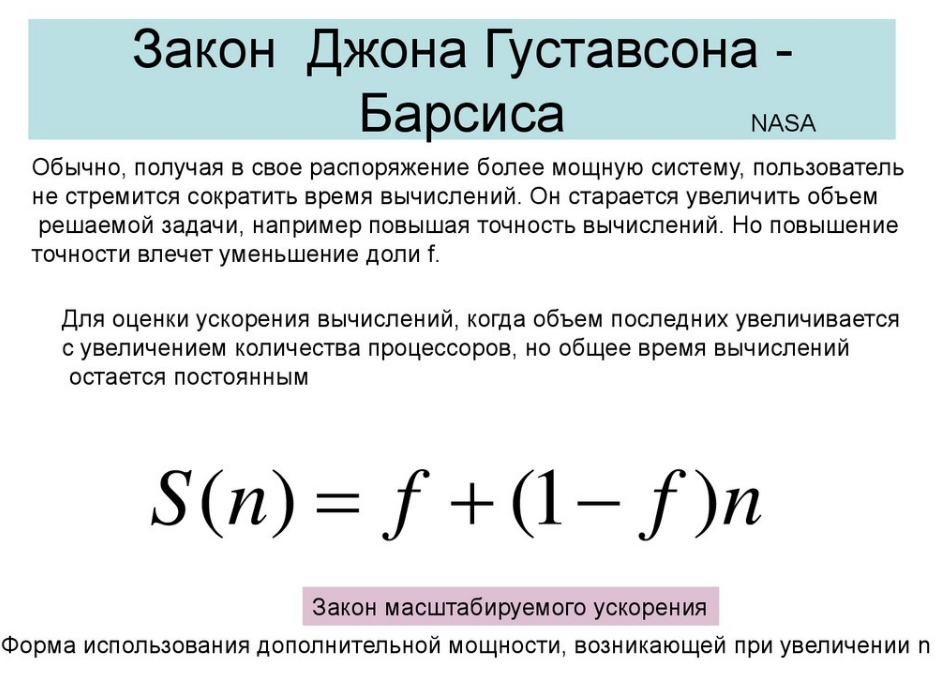
Мультикомпьютеры весьма разнообразны по форме и размерам и плохо поддаются систематизации. Можно выделить два основных варианта: массово-параллельные (МРР) системы и кластеры.

К категории массово-параллельных систем относят дорогостоящие суперкомпьютеры, где компоненты тесно интегрированы за счет специализированного коммуникационного оборудования. Более популярны кластеры, основу которых составляют общедоступные вычислительные и коммуникационные компоненты. В отличие от массово-параллельных систем, кластеры можно сформировать, используя серийное оборудование — серийные сервера, установленные в стандартные стойки, серийное (возможно, дорогое) коммутационное оборудование. Массовость использованного оборудования позволяет достичь демократичной цены. Важным является и то, что отсутствие закрытых технологий позволяет использовать стандартные операционные системы со стандартными средствами разработки.

Кластеры можно отнести к распределенным системам, особенно так называемые децентрализованные кластеры, которые состоят из автономных вычислительных узлов, связанных локальной сетью.

**96. Законы Амдала и Густафсона–Барсиса.**





**97. Особенности CISC, RISC и VLIW (EPIC) архитектур микропроцессоров.**

CISC (Complicated Instruction Set Computer — использующий полный набор команд) — концепция проектирования процессоров, которая характеризуется следующим набором свойств:

- нефиксированное значение длины команды;

- арифметические действия кодируются в одной команде;

- небольшое число регистров, каждый из которых выполняет строго определённую функцию.

RISC (Reduced Instruction Set Computer). Архитектура с сокращённым набором команд.

Характерные особенности RISC -процессоров:

Одинаковая длина команд (упрощает выборку инструкций из памяти);

Использование большого количество регистров, соответственного типа архитектуры (регистровая, ортоганальнорегистровая) - снижает использование ОП;

2-3 способа адресации, в основном регистровая.

Устройство управления на жесткой логике.

Сокращённый набор команд - 50-100 команд (позволяет обойтись без схемы микропрограммного управления);

Простые способы адресации памяти (обеспечивает отсутствие сложных вычислений адреса);

Отсутствие совмещённой операции чтения/записи с обработкой данных;

Необходимость соответствующей компиляции программ для повышения эффективности;

Несовместимость с набором команд CISC МП (непереносимость exe. – файлов).

VLIW (Very long instruction word — сверхдлинное командное слово) - архитектура процессоров с несколькими вычислительными устройствами.

Архитектура процессоров с явно выраженным параллелизмом вычислений, заложенным в систему команд процессора. Являются основой для архитектуры EPIC. Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно. Задача распределения между ними работы решается программно. Загрузкой вычислительных устройств для VLIW-процессора занимается компилятор, на что отводится существенно больше времени (качество загрузки и, соответственно, производительность теоретически должны быть выше).

**98. Архитектура x86. Система команд микропроцессора x86.**

x86 (Intel 80x86) — это общее название семейства микропроцессоров, как разработанных и выпускаемых компанией Intel, так и совместимых с ними процессоров других производителей (AMD, VIA, Transmeta, Winchip и т. д.).

x86 — это CISC-архитектура. Доступ к памяти происходит по «словам». «Слова» размещаются по принципу little-endian. Современные процессоры включают в себя декодеры команд x86 для преобразования их в упрощённый внутренний формат с последующим их выполнением, тем самым они являются RISC и CISC одновременно.

**99. Реальный и защищенный режимы работы микропроцессора x86.**

Реальный режим

В реальном режиме при вычислении линейного адреса, по которому процессор собирается читать содержимое памяти или писать в неё, сегментная часть адреса умножается на 16 (или, то же самое, что и сдвиг влево на 4 бита) и суммируется со смещением (если процессору передаётся не полный адрес из двух 16-битных значений — сегмента и смещения, — а только 16-битное смещение, то сегмент берётся из одного из сегментных регистров). Таким образом, адреса 0400h:0001h и 0000h:4001h ссылаются на один и тот же физический адрес, так как 400h×16+1 = 0×16+4001h.

Такой способ вычисления физического адреса позволяет адресовать 1 Мб + 64 Кб − 16 байт памяти (диапазон адресов 0000h…10FFEFh). Однако в процессорах 8086/8088 всего 20 адресных линий, поэтому реально доступен только 1 мегабайт (диапазон адресов 0000h…FFFFFh), а при адресации выше (в диапазоне 100000h…10FFEFh) происходит «заворот» — старший единичный бит адреса игнорируется и происходит обращение к 64 килобайтам в начальных адресах (0000h…FFEFh).

Процессор 80286 имеет 24-битную адресную шину (возможна адресация 224 = 16 Мб памяти), поэтому в них переполнения не происходит. Компьютеры IBM PC/AT построены на процессоре Intel 80286, но, из соображений совместимости с IBM PC и IBM PC/XT, построенных на Intel 808x, в них был введён логический элемент (вентиль), управляющий работой 21-го адресного провода (A20). Этот логический элемент, получивший название "Gate A20", по умолчанию отключен, что соответствует режиму совместимости, но управляется через контроллер клавиатуры (микросхема Intel 8042).

Защищенный режим

В дополнение к реальному режиме Intel 80286 поддерживает защищенный режим, расширение адресацией физической памяти до 16 МБ и адресуемой виртуальной памяти до 1 Гб, а также предоставление защищенной памяти, которая предотвращает программы от разлагающего друг от друга. Это делается с помощью сегментных регистров только для хранения индекса в таблице дескрипторов, которая хранится в памяти. Есть две такие таблицы, тем Глобальная таблица дескрипторов (GDT) и таблица дескрипторов Local (LDT), каждая из которых содержит до 8192 дескрипторов сегментов, каждый сегмент дает доступ к 64 Кбайт памяти. В 80286, дескриптор сегмента обеспечивает 24-битную в базовый адрес , и этот базовый адрес добавляется к 16-битным смещением , чтобы создать абсолютный адрес. Базовый адрес из таблицы выполняет ту же роль , что буквальное значение регистра сегмента выполняет в режиме реального времени; сегментные регистры были преобразованы из прямых регистров косвенных регистров. Каждый сегмент может быть назначен один из четырех кольцевых уровней , используемых для аппаратной компьютерной безопасности . Каждый дескриптор сегмента содержит также предельное поле сегмента, который определяет максимальное смещение , которое может быть использовано с сегментом. Поскольку Смещения 16 бит, сегменты по - прежнему ограничены до 64 Кбайт каждый в 80286 защищенном режиме.

Суть защищённого режима в следующем: программист и разрабатываемые им программы используют логическое адресное пространство, размер которого может составлять 1 гигабайт. Логический адрес преобразуется в физический адрес автоматически с помощью схемы управления памятью (MMU). При этом содержимое сегментного регистра не связано напрямую с физическим адресом, а является номером сегмента в соответствующей таблице. Благодаря защищенному режиму, в памяти может храниться только та часть программы, которая необходима в данный момент, а остальная часть может храниться во внешней памяти (например, на жестком диске). В случае обращения к той части программы, которой нет в памяти в данный момент, операционная система может приостановить программу, загрузить требуемую секцию кода из внешней памяти и возобновить выполнение программы. Следовательно, становятся допустимыми программы, размер которых больше объема имеющейся памяти, и пользователю кажется, что он работает с большей памятью, чем на самом деле.

Физический адрес формируется следующим образом. В сегментных регистрах хранится селектор, содержащий индекс дескриптора в таблице дескрипторов (13 бит), 1 бит, определяющий к какой таблице дескрипторов будет производиться обращение (к локальной или к глобальной) и 2 бита запрашиваемого уровня привилегий. Далее происходит обращение к соответствующей таблице дескрипторов и соответствующему дескриптору, который содержит начальный 24-битный адрес сегмента, размер сегмента и права доступа, после чего вычисляется необходимый физический адрес путём сложения адреса сегмента со смещением из 16-разрядного регистра.

**100. Аппаратные средства поддержки привилегированного режима микропроцессора x86.**

Средства поддержки привилегированного режима обычно основаны на системном регистре процессора, часто называемом «словом состояния» машины или процессора. Этот регистр содержит некоторые признаки, определяющие режимы работы процессора, в том числе и признак текущего режима привилегий. Смена режима привилегий выполняется за счет изменения слова состояния машины в результате прерывания или выполнения привилегированной команды. Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро-пользователь) или четыре (например, ядро- супервизор- выполнение- пользователь у платформы VAX или 0-1-2-3 у процессоров Intel x86/Pentium).

В обязанности средств поддержки привилегированного режима входит выполнение проверки допустимости выполнения активной программой инструкций процессора при текущем уровне привилегированности.